Design Document

Manikanta Illuri

The design and Ideas for this assignment were derived from TA and tutor help as well as some aspects from my previous quarters work on a similar assignment.

INITIAL Design

Objective:

Deliverables:

- Trie.c
- Trie.h
- Word.c
- Word.h
- Io.c
- Io.h
- Encode.c
- Decode.c
- Code.h
- endian.h
- Makefile
- DESIGN.pdf
- WRITEUP.pdf
- README.md

Trie.c:

Objective: Constructor for a TrieNode. The node's code is set to code. Make sure each of the children node pointers are NULL

TrieNode *trie_node_create(uint16_t index){

- I will use "malloc" to allocate memory for the pointer node.
- Will have an that checks if node if NULL or not
    - If node is not NULL then we will do node ->code = index
- Else
    - Will set node to NULL
- Will return node

}

Objective: Destructor for a TrieNode.

void trie_node_delete(TrieNode *n){

- Will have a if that will check if node is NULL or not

- If node is not NULL then will use free() to free
- Will set node to NULL

}

Objective: Initializes a trie: a root TrieNode with the code EMPTY_CODE. Returns the root, a TrieNode *, if successful, NULL otherwise.

TrieNode *trie_create(void){

- Will call trie_node_create function and pass in EMPYTY_CODE

}

Objective: Resets a trie to just the root TrieNode.

void trie_reset(TrieNode *root){

- Will have an if that checks if to check if node is NULL or not
    - If node is not NULL then will initiate a while loop that run while i is less than ALPHABET
        - Within the loop will check if children index is NULL or not
            - If not will call trie_delete(index val)
            - Will set children index to NULL
            - I++
    - Will return;

}

Objective: Deletes a sub-trie starting from the tree rooted at node n.

void trie_delete(TrieNode *n){

- Will have an if check node is NULL or not
    - Will have a while loop that runs while i is less than ALPHABET
        - Within loop will call trie_node_delete and pass in children index
        - Will do i++
    - Will call trie_node_delete and pass in node
    - Will set node to NULL
    - Will return

}

Objective: Returns a pointer to the child node representing the symbol sym

TrieNode *trie_step(TrieNode *n, uint8_t sym){

- Will have an if to check if children[nym] is NULL
    - If NULL, will return NULL
- Else
    - Will return children[nym]

}

Word.c:

Objective: Constructor for a word where sysms is the array of symbols a Word represents.

Word *word_create(uint8_t *syms, uint32_t len){

- I will use malloc to allocate memory for the word stack and set the top to null
- I used a if to check if the word was NULL or not
    - If not i set word to sym and called malloc to allocate memory
    - Then i set word to len for len
    - Then I will have aif to check if(wordin syms)
        - If so then using while loop that will run while i less than len
            - I will do word->syms[i] = syms[i];

}

Objective: Constructs a new Word from the specified Word, w, appended with a symbol, sym.

Word *word_append_sym(Word *w, uint8_t sym){

- *I will use a loop to append the sym to Word. Thai loop will have a check before it to see if the Word is NULL or not.*
- *I called on calloc to allocate the memory*
    - *Then i set new_append->len to w-len +1*
        - *The in a if i checked if w->len*
            - *I used memcpy to block memory from one location to another. This was a tip given to me during Ta hours.*
        - *I will return the new_append.*

}

Objective: Destructor for a Word, w.

void word_delete(Word *w);{

- *I will check if word is NULL using an if*
    - *Within the if I will use free to free up word*
    - *And set it to null*

}

Objective: Creates a new WordTable, which is an array of Words.

WordTable *wt_create(void){

- *I will use malloc to allocate memory for the word table.*
- *Allocate memory using calloc by using MAX_CODE*
- *While if wt is not NUll I will*
    - *Use a while loop to set it to NULL*

- *While wt is NULL*
    - *I will use a while loop to do word_delete*

}

Objective: Resets a WordTable, wt, to contain just the empty Word.

void wt_reset(WordTable *wt){

- *I will use a loop to and call free to reset the word table to NULL*
- *I will use a while loop to check if i < max)_code*
    - *I will chekc if its nULl*
        - *Then I will do word delete*

}

Objective: Deletes a WordTable, wt, and frees it.

void wt_delete(WordTable *wt){

- *I will use a if to check if the word table is NUll*
    - *If not I will run loop through the table and set the called sym to NULL*
- *I will use a while loop that run while i is less than max_code*
- *Then i will use word delete to delete them.*

}

Io.c:

*** I initiated 6 global variables sto store the values of buffer, index for sym, bit index, bit buffer, total sym, and total bits. This method helped me easily access the variables and append to them easily rather than redeclaring them through the program locally. ***

Objective: This will be a useful helper function to perform reads

int read_bytes(int infile, uint8_t *buf, int to_read){

- *I will use a while loop that checks for if bytes is equal to and if fnl is already equal to to_read.*
    - *Within loop i will set bytes to read(and passing infile buf and to_read)*
- *I will return fnl*

}

Objective: This function is very much the same as read_bytes(), except that it is for looping calls to write().

int write_bytes(int outfile, uint8_t *buf, int to_write){

- *I will use a while loop that checks for if bytes is equal to and if fnl is already equal to to_read.*
    - *Within loop i will set bytes to write(and passing infile buf and to_read)*

- *I will return fnl*

}

Objective: This reads in sizeof(FileHeader) bytes from the input file.

void read_header(int infile, FileHeader *header){

- *I will run a if while little_endian is not true*
  - *Within the if I will swap32 header and magic*
  - *I will swap 16 header and protection.*
- *I will call read-bytes and pass in infile, (uint8_t *) header, sizeof(FileHeader)*

}

Objective: Writes sizeof(FileHeader) bytes to the output file.

void write_header(int outfile, FileHeader *header){

- *I will run a if while little_endian is not true*
  - *Within the if I will swap32 header and magic*
  - *I will swap 16 header and protection.*
- *I will call write-bytes and pass in infile, (uint8_t *) header, sizeof(FileHeader)*

}

Objective: An index keeps track of the currently read symbol in the buffer.

bool read_sym(int infile, uint8_t *sym){

- *I will have an if that will check if sym_index_val is !*
  - *I will read the byte inside using read_bytes*
- *Outside the if will have another if that will add one to finish and read_b*
- *I will call the sym to use the sym_index_val = sym_index_val_val % BLOCK; on the pointer*

}

Objective: "Writes" a pair to outfile. In reality, the pair is buffered.

void write_pair(int outfile, uint16_t code, uint8_t sym, int bitlen){

- *I will have a while loop that will run while the iterator is less than bitlen*
  - *Within the loop I will have an if that will run while code >> (i % 16) & 1*
    - *Within the if i will bit_buffer[bit_index_val / 8] |= (1 << (bit_index % 8));*
- *Outside the loop and if have another loop for the symbol.*

-within this loop Will increase the bit buffer using the bit_buffer[bit_index_val / 8] |= (1 << (bit_index % 8));

*I will have if block that will check if bit_index_val == 8*index.*

- *I will write bytes.*

}

Objective: Writes out any remaining pairs of symbols and codes to the output file.

void flush_pairs(int outfile){

- *Will have a for loop to print the pairs and I will have conduction for the loop r run will i is less than Block - it_index.*
    - *Within I will call the buffer.*
- *Outside the loop I will write_bytes.*

}

Objective: "Reads" a pair (code and symbol) from the input file.

bool read_pair(int infile, uint16_t *code, uint8_t *sym, int bitlen){

- *I will have a while loop that will run while the iterator is less than bitlen*
    - *Within the loop I will have an if that will run while code >> (i % 16) & 1*
        - *Within the if i will bit_buffer[bit_index_val / 8] |= (1 << (bit_index % 8));*
- *Outside the loop and if have another loop for the symbol.*

-within this loop Will increase the bit buffer using the bit_buffer[bit_index_val / 8] |= (1 << (bit_index % 8));

*I will have if block that will check if bit_index_val == 8*index.*

- *I will read bytes.*

}

Objective: "Writes" a pair to the output file.

void write_word(int outfile, Word *w){

- *Will write using for loop the outfile using fprintf and writing permissions. I will check if the file is NULL.*
- *I ran a if while w is not NULL*
    - *I will return exit(EXIT_FAILUR)*
- *Else*
    - *I will run a while loop*
        - *This loop will run while while (i < w->len)*
            - *Within loop will have a if that will check if the sym index val is == BLOCL*
                - *If so wil wire bites*
            - *Will fo sym_buffer*
                - *Increase iterator*

}

Objective: Writes out any remaining symbols in the buffer to the outfile.
void flush_words(int outfile){

- *Will have an if statement to print the pairs and I will have conduction for the if run will be less than Block - it_index.*
    - *Within I will call the buffer.*
- *Outside the loop I will write_bytes.*

}

Encoded.c:

Objective: Contains the main() function for the encode program.

I will use switch cases to address all the flags that can be imputed for this program file. If verbose is true I will enable the encoded function and print the stats.
Your encode program must support the following getopt() options:
• -v : Print compression statistics to stderr.
• -i : Specify input to compress (stdin by default)
• -o : Specify output of compressed input (stdout by default)
The program was implemented according to the pseudo code provided by the asgn document. The program will first read the symbols in once they have been read it will write the final code to the out file. If verbose is enabled I will enable the printing of the compression statistics.

Decoded.c:

Objective: Contains the main() function for the decode program.

I will use switch cases to address all the flags that can be imputed for this program file. If verbose is true I will enable the decode function and print the stats.
Your decode program must support the following getopt() options:
• -v : Print decompression statistics to stderr.
• -i : Specify input to decompress (stdin by default)
• -o : Specify output of decompressed input (stdout by default)

The program was implemented according to the pseudo code provided by the asgn document. The program will first read the symbols in once they have been read it will write the final code to the out file. If verbose is enabled I will enable the printing of the compression statistics.