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: The objective of this assignment is to create a program that can encrypt and decrypt a given input. For this a ss ignment I will be using keygen to produce an ss public and private key pair and use the encrypt and decrypt files to manipulate the input.

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

}

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.*

}

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.*

}

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*

}

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*

}

Io.c:

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 the fscanf to read the bytes. I will use if to check if the word table is null first.*

}

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 fprintf to print out the output.*

}

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

void read_header(int infile, FileHeader *header){

- *I will use the scanf to read the header. I will check if the header is NULL first*

}

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

void write_header(int outfile, FileHeader *header){

- *I will write the file header as the outfile using the writing permissions.*

}

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

bool read_sym(int infile, uint8_t *sym){

- *Will read using fscanf to read the infule and the sym in the inf file. WIll have checked to see if the infile is set to NULL.*

}

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){

- *Will will take the outfile with write permission and write to it using fprintf.*

}

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){

- *Will read the pairs using fscanf*

}

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.*

}

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

}

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.

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.