Nicolas Ayala
nmayala@ucsc.edu

Design of project

The project has the following codes, defined in code.h

```
# define STOP_CODE 0 // Signals end of decoding / decoding .
# define EMPTY_CODE 1 // Code denoting the empty Word .
# define START_CODE 2 // Starting code of new Words .
# define MAX_CODE UINT16_MAX
```

There will be a trie.h and a trie.c, an implementation of the trie ADT.

A trie node is the following structure:

```
struct TrieNode{
        TrieNode *children[256];  //array of possible next letters
        uint16_t code;
}

TrieNode *trie_node_create(uint16_t code){
        TrieNode *fresh = malloc(sizeof(TrieNode));
        fresh->children = calloc(256, sizeof(TrieNode *));
        fresh->code = code;
        return code;
}

void trie_node_delete ( TrieNode *n){
        free(n->children);
        free(n);
}

//Creates a root node with EMPTY_CODE as code
TrieNode * trie_create ( void ){
        TrieNode *root = trie_node_create(EMPTYCODE);
        return root;
}

//Resets a Trie back to its root.
void trie_reset(TrieNode *root){
        for(int i = 0; i < 256; i++){
                if(root->children[i])
                        trie_delete(root->children[i]);
```

```
        }
    }

    //Deletes a Trie starting at the root
    void trie_delete ( TrieNode *n){
        for(int i = 0; i < 256; i++){
            if(root->children[i])
                trie_delete(root->children[i]);
        }
        Trie_node_delete(n);
    }

    //Returns child corresponding to symbol sym, if it exists
    TrieNode *trie_step ( TrieNode *n, uint8_t sym ) {
        return n->children[sym];
    }
```

There will be a word.h and a word.c, an implementation of a Word table, an array of strings

```
    //A word is the following structure
    struct Word{
        uint8_t *syms;  //Array of symbols, in this case ASCII characters
        uint32_t len;     //length of array of symbols
    }

    //A WordTable is an array of words
    typedef  Word*  WordTable;

    Word * word_create ( uint8_t *syms , uint32_t len ) {
        Word *fresh = malloc(sizeof(Word));
        fresh -> syms = malloc(len);
        strcpy(fresh->syms, syms); //copy string into struct
        fresh->len = len;
        return fresh;
    }


    //Creates a new word which is the old word plus an appended symbol
    Word * word_append_sym ( Word *w, uint8_t sym ) {
        Word *new_word = word_create(w->syms, w->len + 1);
        new_word->syms[w->len] = sym;
        return new_word
    }
```

```
void word_delete ( Word *w){
        free(w->syms);
        free(w);
}

//Creates a word table of size MAX_CODE, so as to fit all possible codes
//New word table will already contain empty word at EMPTY_CODE
WordTable * wt_create ( void ){
        WordTable *fresh = calloc(MAX_CODE, sizeof(Word));
        Word *empty = word_create("",0);
        fresh[EMPTY_CODE] = empty;
        return fresh;
}

//Resets a word table to just the empty word
void wt_reset ( WordTable *wt){
        for(int i = 0;i<MAX_CODE && wt[i];i++){
                word_delete(wt[i]);
        }
        Word *empty = word_create("",0);
        wt[EMPTY_CODE] = empty;
}

void wt_delete ( WordTable *wt){
        for(int i = 0;i<MAX_CODE && wt[i];i++){
                word_delete(wt[i]);
        }
        free(wt);
}
```

There will be an endian.h, code that deals with endianness

Theres a function that tests the endianness of a computer

```
bool   is_big(void);
bool   is_little(void);
```

There are functions that swap the endianness of a 16, 32, and 64 bit datum

```
uint16_t   swap16(uint16_t)
uint32_t   swap32(uint32_t)
```

```
        uint64_t   swap64(uint64_t)
```

There will be an io.h and an io.c

```
    #define   MAGIC_NUMBER    0x8badbeef

    #define   BLOCK                 4096;

    struct FileHeader {
        uint32_t magic ;
        uint16_t protection ;
     };


    //inFile is file descriptor
    //header is pointer to FileHeader to write to
    //
    //Reads a header from a file and places it into *header
    void read_header (int infile , FileHeader * header ){
        read(infile, header, sizeof(FileHeader));

        if(is_big()){
            header->magic = swap32(header->magic);
            header->protection = swap16(header->protection);
        }
    }

    //Writes header to output file
    void write_header (int outfile, Fileheader *header){
        if(is_big()){
            header->magic = swap32(header->magic);
            header->protection = swap16(header->protection);
        }

        write(outfile, header ,sizeof(header));
     }

    uint8_t     pair_buffer[BLOCK_SIZE];
    uint16_t   pair_bit_index = 0;
    uint16_t     pair_end = 0;


    uint8_t     char_buffer[BLOCK_SIZE];
    uint16_t   char_byte_index = 0;
```

```
uint16_t     char_end = 0;


//reads a symbol from the infile
bool read_sym (int infile , uint8_t * sym ){
        if(char_byte_index >= char_end){
                char_end = read(infile, char_buffer, BLOCK_SIZE);
                char_byte_index = 0;
        }
        if(char_end == 0){
                return false;
        }
        *sym = char_buffer[char_byte_index];
        char_byte_index++;
        return true;
}

//Outputs a pair to the outfile
void buffer_pair (int outfile , uint16_t code , uint8_t sym , uint8_t bitlen ) {

        uint32_t pair = code + (sym << bitlen);

        for(int i = 0; i < bitlen + 8; i ++){
                uint16_t bit = (pair_bit_index + i) % (BLOCK * 8);
                pair_buffer[bit/8] &= ~(1 << bit % 8 )                  //clearing bit
                pair_buffer[bit/8] += ((pair >> i) % 2)<< (bit %8); //replacing bit

                if(pair_bit_index + i == BLOCK * 8 - 1){              //if we just filled it, write
                        write out block
                }
        }

        pair_bit_index += bitlen + 8;
        pair_bit_index %= BLOCK * 8;
}

// Writes out any remaining pairs of symbols and codes to the output file .
//
// outfile : File descriptor of the output file to write to.
// returns : Void .
//
void flush_pairs (int outfile){
        int bytes = (pair_bit_index)/8+1;
```

```
        if(pair_bit_index % 8 == 0){
                bytes--;
        }
        write out up to bytes
        free(pair_buffer);
}


//Reads in a pair
bool read_pair (int infile , uint16_t *code , uint8_t *sym , uint8_t bitlen ) {
        uint32_t pair = 0;

        for(int i = 0; i < bitlen + 8; i++){
                if(pair_bit_index + i == pair_end * 8){
                        read next block, return false if empty;
                }

                uint16_t bit = (pair_bit_index + i) % (BLOCK * 8);
                pair += (pair_buffer[bit / 8] >> (bit % 8)) % 2) << i;
        }

        if(pair_bit_index == 0){
                read next block, return false if empty;
        }

        *code = (uint16_t) (pair & ~( ~0 << bitlen));
        *sym = (uint8_t) (pair >> bitlen);

        return (*code != STOP_CODE);
}

//Writes out a word
void buffer_word (int outfile , Word *w){
        for( int i = 0; i < w->len; i++){
                char_buffer[(i + char_byte_index) % BLOCK] = w->syms[i];

                if( i + char_byte_index == BLOCK -1){   //If we just filled block, write
                        write out block
                }
        }

        char_byte_index += w->len;
        char_byte_index %= BLOCK;
```

```
}

//Writes out remaining words
void flush_words (int outfile ){
        if(char_byte_index == 0){
                return;
        }

        write out up to char_byte_index
}
```

The project will have two executables, whose mains are found in encode.c and decode.c
Pseudo code for encode is as follows, and is written by Darrel Long:

```
root = TRIE_CREATE()
curr_node = root
prev_node = NULL
curr_sym = 0
prev_sym = 0
next_code = START_CODE
while READ_SYM(infile, &curr_sym) is TRUE
  next_node = TRIE_STEP(curr_node, curr_sym)
  if next_node is not NULL
    prev_node = curr_node
    curr_node = next_node
  else
    BUFFER_PAIR(outfile, curr_node.code, curr_sym, BIT-LENGTH(next_code))
    curr_node.children[curr_sym] = TRIE_NODE_CREATE(next_code)
    curr_node = root
    next_code = next_code + 1
 if next_code is MAX_CODE
    TRIE_RESET(root)
    curr_node = root
    next_code = START_CODE
    prev_sym = curr_sym
if curr_node is not root
  BUFFER_PAIR(outfile, prev_node.code, prev_sym, BIT-LENGTH(next_code))
  next_code = (next_code +1) % MAX_CODE
  BUFFER_PAIR(outfile, STOP_CODE, 0, BIT-LENGTH(next_code))
FLUSH_PAIRS(outfile)
```

Pseudo code for decode is as follows, and is written by Darrel Long:

```
table = WT_CREATE()
curr_sym = 0
```

```
curr_code = 0
next_code = START_CODE
while READ_PAIR(infile, &curr_code, &curr_sym, BIT-LENGTH(next_code)) is TRUE
  table[next_code] = WORD_APPEND_SYM(table[curr_code], curr_sym)
  buffer_word(outfile, table[next_code])
  next_code = next_code + 1
  if next_code is MAX_CODE
    WT_RESET(table)
    next_code = START_CODE
FLUSH_WORDS(outfile)
```