Design of Lempel-Ziv Compression

1. Pseudo-code for Word

```
// create a word with symbols
WORD_CREATE(symbols)
    word = MALLOC()
    word.symbols = MAKE_COPY(symbols)
    RETURN word

// append a symbol to word
WORD_APPEND_SYM(word, symbol)
    new_word = MALLOC
    new_word.symbols = MAKE_COPY(word.symbol)
    APPEND(new_word.symbols, symbol)
    RERTURN new_word

// delete a word
WORD_DELETE(word)
    DELETE(word.symbols)
    DELETE(word)

// create a word table
WT_CREATE()
    word_table = MALLOC()
    word_table[EMPTY_CODE] = WORD_CREATE(NULL)
    RETURN word_table

// reset a word table
WT_RESET(word_table)
    for word in word_table
        WORD_DELETE(word)
```

2. Pseudo-code for TRIE

```
// create a trie node
TRIE_NODE_CREATE(index)
    node = MALLOC()
    node->code = index
    RETURN node
```

```
// create root trie node
TRIE_CREATE()
    root = TRIE_NODE_CREATE(EMPTY_CODE)
    RETURN root

// reset a trie node
TRIE_RESET(root)
    for node in root.children
        TRIE_RESET(node)

// get the child node according to symbol
TRIE_STEP(node, symbol)
     RETURN node.children[symbol]
```

3. Pseudo-code for IO

```
// read file header from file
READ_HEADER(file)
    bytes = READ_BYTES(file, header_size_bytes)
    header = CAST(bytes)
    RETURN header

// write file header to file
WRITE_HEADER(file, header)
    WRITE_BYTES(file, header, header_size_bytes)

// read a symbol from file
READ_SYM(file)
     // already in buffer
    if symbol_buffer is not empty
        symbol = symbol_buffer[index]
        RETURN symbol
    else
        // buffer empty, read from file to buffer
        symbol_buffer = READ(file, 4096)
        symbol_buffer = symbol_buffer[index]
        RETURN symbol


// buffers a pair. A pair is comprised of a symbol and an index.
BUFFER_PAIR(file, code, symbol, bit_length)
     // construct pair bits
    data = PACK(symbol, code)
    // pack data to bits buffer
```

```
    PACK_BIT(bits_buffer, data)
    if data is all packed
        RETURN

    // bits buffer is full, need to write
    WRITE_BYTES(file, bits_buffer, 4096)
    // pack left bits of data
    PACK_BIT(bits_buffer, data)


// flush bit buffer to file
FLUSH_PAIR(file)
    if bits_buffer is not empty
        WRITE_BYTES(file, bits_buffer, left_byte_size)

// read pair from file
READ_PAIR(file)
     pair = UNPACK_BITS(bits_buffer)
     // pair is done
     if pair is unpacked
         return pair

     // there is left bits in file
     bits_buffer = READ_BYTES(file, 4096)
     // no data any more
     if bits_buffer is empty
         RETURN NULL

     // unpack left
     pair = UNPACK_BITS(bits_buffer)
     RETURN pair

// buffer a word
BUFFER_WORD(file, word)
    BUFFER(symbol_buffer, word.symbols)
    if all buffered
      RETURN
    // buffer is full
    WRITE_BYTES(file, symbol_buffer, 4096)
    // buffer left symbols
    BUFFER(symbol_buffer, word.symbols)

// flush words to file
FLUSH_WORDS(file)
```

```
    if symbol_buffer is not empty
        WRITE_BYTES(file, symbol_buffer, bytes_left)
```

4. Pseudo-code for compress and decompress

Use the LZ78 Algorithm Pseudo-code from assignment document.

```
COMPRESS(infile, outfile)

    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)

DECOMPRESS(infile, outfile)
        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)
```