Assignment 6 Design

Caitlin Smith

March 13, 2023

Program Description

In this assignment, two programs perform LZ78 compression and decompression. All of the functions for the program are within the trie.c, word.c, and io.c files. There are separate programs for compression and decompression. Those two programs utilize command-line arguments in the main function in order to set specifications. There are two other given .h files (code.h and endian.h) that are used throughout the program.

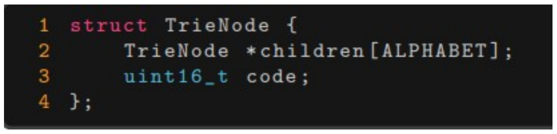
1 trie.c

Purpose: This file contains the functions for creating and manipulating trie structures. The interface is given in the trie.h file.

Pseudocode:

Include standard .h files and trie.h.

Define the TrieNode ADT as shown below.



```
struct TrieNode {
    TrieNode *children[ALPHABET];
    uint16_t code;
};
```

trie_node_create: Create a function that constructs a TrieNode and returns it.
Allocate memory for a new TrieNode.
Set the new TrieNode's code to the passed code variable.
Set the children node pointers to NULL.
Return the TrieNode.

trie_node_delete: Create a void function that destructs a TrieNode.

```
Free and null the passed pointer.
```

```
trie_create: Create a function to initialize a trie. It returns a TrieNode.
Create a TrieNode pointer for the root and set the code to EMPTY_CODE.
If successful, return the root TrieNode pointer.
If not, return NULL.
```

```
trie_reset: Create a void function to reset a trie to just the root node.
Loop through the children.
    If it is not NULL,
        Call the delete function on it.
        Set to NULL.
```

```
trie_delete: Create a void function to delete a sub-trie started at the passed node n.
Loop through the children of the passed node.
    Call trie_delete, passing the child.
Call trie_node_delete on the passed node.
```

```
tri_step: Create a function to return the child node with the symbol specified in the parameters.
If the symbol exists, return the node of that symbol.
If not, return NULL.
```

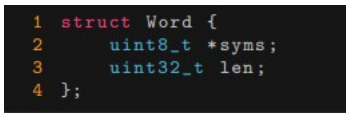
2 word.c

Purpose: This file contains the functions for creating and manipulating word table structures. The interface is given in the word.h file.

Pseudocode:

Include standard .h files and word.h.

Define the Word ADT as shown below.



```c
struct Word {
    uint8_t *syms;
    uint32_t len;
};
```

A WordTable is defined as an array of Words.

```
word_create: Create a function that constructs a Word and returns it.
Allocate memory for a word.
```

Set the new Words's len to the passed len variable.
Allocate memory for the syms array and copy it from the passed syms.
Return the Word.

word_append_sym: Create a function that constructs a new Word from the passed Word and appends the passed symbol.
Create a new Word variable with the passed Word's len plus 1.
Copy the passed Word's syms into the new one.
Set the last syms in the new Word to the passed sym.
Return the newly created Word.

word_delete: Create a void function that destructs a Word.
Free and null the passed Word.

wt_create: Create a function that creates a new WordTable.
Allocate for a WordTable array of size MAX_CODE.
At the index EMPTY_CODE initialize a single empty Word.

wt_reset: Create a void function to reset a WordTable.
Iterate and delete each of the Words, leaving the first.
Set to NULL.

wt_delete: Create a void function to delete all words and free.
Call wt_reset.
Call word_delete on the index 0.
Free and set to NULL.

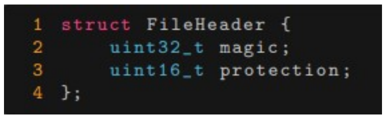
3 io.c

Purpose: This file contains the functions for reading and writing the in and out files. The interface is given in the io.h file.

Pseudocode:

Include standard .h files and io.h.

The FileHeader ADT is defined as shown below.


```
struct FileHeader {
    uint32_t magic;
    uint16_t protection;
};
```

Set the total_syms and total_bits extern variables to 0.

```
read_bytes: Create a function that reads from the passed infile.
Create a variable to count the number of bytes read.
Loop until all the bytes have been read.
    Call read() on the infile.
    Increase the count variable by 1.
Return the count of bytes read.
```

```
write_bytes: Create a function that writes to the passed outfile.
Create a variable to count the number of bytes written.
Loop until all the bytes have been written.
    Call write() on the outfile.
    Increase the count variable by 1.
Return the count of bytes written.
```

read_header: Create a function to read the FileHeader.
Call the function that reads bytes and store the output to a variable.
If it is not little endian, swap the endian of the magic and protection.
Check if the magic number is correct.

write_header: Create a function to write from the FileHeader.
If it is not little endian, swap the endian of magic and protection.
Call the function that writes the bytes with the header.

Create a symbol buffer, symbol position, pair buffer and pair position as global variables.

read_sym: Create a function to read symbols.
If the buffer is full, read more bytes into it. Reset the symbol position.
Set the symbol to what is current in the buffer and increment the position variable.
Return true.

write_pair: Create a function to write a pair to the outfile.
Starting with the LSB, buffer the bits of code with a for loop from 0 to bitlen. If the buffer is full, flush.
Starting with the LSB, buffer the bites of the symbol with a for loop from 0 to 8. If the buffer is full, flush.

flush_pairs: Create a function to write remaining pairs to the outfile.
If there are any pairs that did not make it into the buffer, write to the outfile. Reset the buffer and position tracker.

read_pair: Create a function that reads a pair from the infile.
Create an index variable to track the current bit.
The code is put in the code pointer with a for loop from 0 to bitlen. If the buffer is full, read in more and reset the position.
The symbol is put in the symbol pointer with a for loop from 0 to 8. If the buffer is full, read in more and reset the position.
If the read code is STOP_CODE there are no more pairs to be read, return false.
Otherwise, return true.

write_word: Create a function that writes a pair to the output.
Put each symbol into a buffer.
If the buffer is filled, flush to the outfile.

flush_words: Create a function to write any remaining symbols to the outfile.
If there are any symbols that are left, write to the outfile.

4 encode.c

Purpose: This file contains the encode main function. It utilizes command-line arguments in order to set specifications. It compresses the infile to the outfile.

Pseudocode:

Define OPTIONS to be "vi:o:h".

The following will all be within the main().

Initialize int opt.

Create variables for the command line options with arguments. The defaults are used if they are not specified in the command-line.

Create a while loop with getopt.

Case v enables statistics output.

Case i sets the input file with default stdin. Open the infile, printing a message if an error occurs.

Case o sets the output file with default stdout.

Case h displays a help message with usage.

Create a FileHeader. Call the read_header function passing the infile.

Open the outfile, printing a message if an error occurs. Make sure the permissions match the read header.

```
    Write the FileHeader to the outfile.
    Create a trie. Also create trackers for the current node and next code.
    Create variables for the previous node and previous symbol.
    While the read symbol function is returning true,
        Set the current symbol to the one being read.
        Set the next node using the trie step function.
        If that next node is not NULL, set the previous to current and current to next.
        Else,
            Call the write pair function with the current code and symbol.
            Set the current node to the root of the trie.
            Increment the value of the next code.
        If the next code is the MAX_CODE,
            Call the trie reset function.
            Set the previous symbol to the current symbol.
    Check if the current node is pointing to the root.
    Call the write pair function.
    Write the pair (STOP_CODE, 0) to mark the end of the compressed output.
    Call the flush pairs function.
    Print the verbose including compressed file size, uncompressed file size, and space saving.
    Close the files.
    Return 0.
```

5 decode.c

Purpose: This file contains the decode main function. It utilizes command-line arguments in order to set specifications. It decompresses the infile to the outfile.

Pseudocode:

```
Define OPTIONS to be "vi:o:h".

The following will all be within the main().
    Initialize int opt.
    Create variables for the command line options with arguments. The defaults are used if they
are not specified in the command-line.

    Create a while loop with getopt.
    Case v enables statistics output.
    Case i sets the input file with default stdin. Open the infile, printing a message if an error
occurs.
    Case o sets the output file with default stdout.
```

Case h displays a help message with usage.

```
Create a FileHeader and call the read header file.
Open the outfile, printing a message if an error occurs. Make sure the permissions match.
Create a new WordTable, setting entries to NULL.
Create variables for the current and next code. Next code is set to START_CODE.
Use a loop and the read pair function to read all pairs in the infile,
    Append the word from the read code wo the symbol. Add this to the table at the index of the next code.
    Use the created function for writing the word.
    Increment the next code variable by 1.
    If the next code variable is equal to MAX_CODE,
        Reset the table.
        Set the next code variable to START_CODE.
Call the flush words function.
Print the verbose including compressed file size, uncompressed file size, and space saving.
Close the files.
Return 0.
```

6 Makefile

Purpose: This is a file that is used to clean, format, and compile the entire program.