Nicolas Corfmat
CSE13S, Winter 2023
ncorfmat@ucsc.edu

Assignment 6 DESIGN.pdf

1 Description of Program:

This project consists of two main programs: *encode* and *decode*, which implement LZ78 compression and decompression respectively. The *encode* program is used to compress either text or binary files while *decode* decompresses compressed files.

2 Files to be included in directory "asgn6":

1. encode.c:
    - Contains the main() function for the encode program.
2. decode.c:
    - Contains the main() function for the decode program.
3. trie.c:
    - C file containing the Trie ADT.
4. word.c
    - C file containing the Word ADT.
5. io.c
    - C file containing the I/O module
6. trie.h
    - Header file for the Trie ADT.
7. word.h
    - Header file for the Word ADT.
8. io.h
    - Header file containing the interface for the I/O module.
9. endian.h
    - Header file for the endianness module.
10. code.h
    - Header file containing macros for reserved codes.
11. Makefile:
    - Supports the compilation of the all .c programs in the "asgn6" directory.

12. README.md:

- Proper explanation of how to run the program and *Makefile*. It also contains a list of available command-line options and explains their functions, as well as any false positives reported by scan-build.

13. DESIGN.pdf:
    - Explains the thought process behind the program with sufficient detail. Comes in PDF format.
14. WRITEUP.pdf:
    - PDF file created using LaTeX that explains concepts learned from this assignment.

3 Pseudocode / Structure:

io.c

Initialize a static sym index tracker to 0
Initialize a static bit index tracker to 0

Initialize a static buffer that holds up to 4096 read bytes
Initialize a static buffer that holds up to 4096 written bytes

GET_BIT(*buf, bits):
    Return the least significant bit at specified index in the buffer

SET_BIT(*buf, bits):
    Set the least significant bit at specified index in the buffer to 1

BITS_TO_BYTES(bits):
    If the number of bits is less than 8:
        Return 1 byte

    Divide bits by 8 and return the rounded down value

READ_BYTES(infile, *buf, to_read):
    Create trackers for total bytes and bytes read, starting at 0

    While the total bytes read is not equal to bytes to read and bytes read is not 0:
        Read “to read” bytes from infile

        If there was a problem reading bytes:
            Print an error message and exit program

Add bytes read to total bytes

Return total bytes read

WRITE_BYTES(outfile, *buf, to_write):
Create trackers for total bytes and bytes read, starting at 0

While the total bytes written is not equal to bytes to write and bytes written is not 0:
Write “to write” bytes in outfile

If there was a problem reading bytes:
Print an error message and exit program

Add bytes read to total bytes

Return total bytes written

READ_HEADER(infile, header):
READ_BYTES(infile, header, size of header struct)

Add size of header struct to total bits counter

Swap endianness if the byte order isn’t little endian. This means swapping the magic number and protection

Verify the magic number and print an error message if the magic number is wrong. Exit program if this is the case.

WRITE_HEADER(outfile, header):

Add size of header struct to total bits counter

Swap endianness if the byte order isn’t little endian. This means swapping the magic number and protection

Verify the magic number and print an error message if the magic number is wrong. Exit program if this is the case.

Write the bytes of header to outfile

```
READ_SYM(infile, *sym):

    If the read buffer is empty:
        Read 4096 bytes from infile into read buffer

        If bytes read exceeds buffer capacity:
            Wrap around index

    Retrieve the next symbol and save it into *sym
    Increment symbol index, making sure not to exceed buffer limit

    If the sym index exceeds buffer size:
        Increment total syms
        Return false

    Return true

WRITE_PAIR(outfile, code, sym, bitlen):

    For bitlen bits of code:
        Write the least significant bit of code into the buffer

        Increment the bit index and total bits

        If the end of the buffer has been reached:
            FLUSH_PAIRS(outfile)
            Reset bit index to 0

    For all 8 bits of sym:
        Write the least significant bit of sym into the buffer

        Increment the bit index and total bits

        If the end of the buffer has been reached:
            FLUSH_PAIRS(outfile)
            Reset bit index to 0
```

```
FLUSH_PAIRS(outfile):
    Write any remaining bytes from write buffer to outfile

READ_PAIR(infile, *code, *sym, bitlen):
    Initialize code and sym to 0

    For bitlen bits of code:
        If we are at the beginning of the buffer:
            READ_BYTES(infile, write_buffer, 4096)

        Read the least significant bit of current input and store it in code

        Increment the bit index

    For all 8 bits of sym:
        If we are at the beginning of the buffer:
            READ_BYTES(infile, write_buffer, 4096)

        Read the least significant bit of current input and store it in sym

        Increment the bit index

    If the code has reached STOP_CODE:
        Return false

    Return true

WRITE_WORD(outfile, *w):
    For every symbol in the given word:
        Write the symbol into outfile

        Increment sym index and total syms

        If the end of the buffer has been reached:
            FLUSH_WORDS(outfile)
            Reset sym index to 0

FLUSH_WORDS(outfile):
    Write out any remaining symbols in the read buffer to outfile
```

trie.c

Create the TrieNode constructor:
    Allocate memory for the TrieNode
    Set the node's code

    For every character in ASCII:
        Set the node's children to NULL

    Return the TrieNode

Create a function that deletes a TrieNode "n":
    Dereference "n" and set it to NULL

Create a function that creates a trie:
    Initalize a root TrieNode with code EMPTY_CODE

    If the root TrieNode is NULL:
        Return NULL

    For every character in ASCII:
        Set the node's children to NULL

    Return the root TrieNode

Create a function that resets a trie to just the root TrieNode:
    For every character in ASCII:
        Delete the child node
        Set the child node to NULL
Create a function that deletes a sub-trie starting from the trie rooted at node n:
    For every character in ASCII:
        If the child node is not NULL:
            Recursively call this delete function on the child node

    Delete the parent trie node
    Set the parent node to NULL

Create a function that returns a pointer to the child node representing a given symbol:
    If the child node pointing to given symbol is NULL:

Return NULL

Return the child node of the given symbol

word.c

Create the Word constructor:
Allocate memory for the Word
Assign the given array of symbols to the Word
Assign the given length of the array of symbols to the Word

Return the Word

Create a function that constructs a new Word from a given Word with the appended symbol:
If given Word is NULL:
Return Null

Create a new Word with the given Word's symbols and a length of +1

If the new Word is NULL:
Return NULL

Append the symbol to the end of the new Word

Return the new Word

Create a function that deletes a word:
Dereference the word
Set the word to NULL

Create a function that creates a WordTable:
Allocate memory for the WordTable, initializing it with an empty Word at index MAX_CODE

If the WordTable is NULL
Return NULL

Allocate memory for the Word at index MAX_CODE

If the Word at index MAX_CODE is NULL:

Return NULL

Set the symbols of the Word at MAX_CODE index to NULL
Set the length of the Word at MAX_CODE index to 0

Return the WordTable

Create a function that resets a given WordTable:
For every Word in WordTable starting at START_CODE and ending at MAX_CODE:
Delete the Word
Set the Word as NULL

Create a function that deletes a WordTable:
Dereference the WordTable
Set the WordTable as NULL

encode.c
Main function:

Initialize current symbol to 0
Initialize previous symbol to 0
Initialize next code to START_CODE
Initialize verbose output to false
Create input and output filestreams
Create a FileHeader variable

While retrieving arguments from command line:
If user types v:
Enable verbose output
If user types i:
Open specified input file to read only
If user types o:
Open specified output file
If user types h:
Display program usage

Obtain the file size and protection bit mask
Write the filled out file header to outfile

Create an empty parent trie and make a copy of it

Create a TrieNode variable to keep track of the previous trie node

While still reading symbols from input file:
next_node = trie_step(curr_node, curr_sym)

If next_nocde is not NULL:
Assign current node to previous node
Assign next node to current node
Else:
Write pair to outfile, using current node's code and current symbol
Set the current node at current symbol to the node at next code
Assign the parent trie to current node

If end of code has been reached:
Reset the parent trie
Assign the parent node to the current node
Reset the next code to the start code

Assign the current symbol to the previous symbol

If the current node does not point to the parent trie:
Write pair to outfile, using previous node's code and previous symbol

Write the pair (STOP_CODE, 0) to signal the end of compressed output (Darrell Long)

Close input and output files

decode.c

Main function:

Initialize current symbol to 0
Initialize current code to 0
Initialize next code to START_CODE
Initialize verbose output to false
Create input and output filestreams
Create a FileHeader variable

While retrieving arguments from command line:
If user types v:

Enable verbose output
If user types i:
Open specified input file to read only
If user types o:
Open specified output file
If user types h:
Display program usage

Read the input file's header and verify the magic number

Create a WordTable

While reading pairs from input file:
Append the read symbol with the word and add it to the WordTable at index next_code

Write the word to outfile

Increment next_code by 1

If next_code has reached MAX_CODE:
Reset the WordTable
Reset next_code to START_CODE

Flush any buffered words
Close input and output files

4 Credits:

- In order to set the magic number and protection to the output file header, I used a version of Professor Long's code which he provided in the "tests" library in the assignment 6 repository.