LAB 7: Lempel Zip Compression

Design Document

Pseudo Code:

TRIE - trie.c

Trie Node Structure:
TrieNode: structure containing an array of size 256 (ALPHABET) and a uint16 unique code.

Creating a Trie Node:
```
trie_node _create(){
TrieNode *trie = (TrieNode)calloc(1, sizeof(trienode));
for (i = 0; i < ALPHABET; i++)
        trie->children[i] = NULL;
   }
trie->code = code;
Return trie;
}
```

English Translation:
Trie node create function takes in a code (a uint16).
  Allocate memory of size trienode for trie using calloc;
  Set all the children to NULL;
  Set the trie code to the code that is passed in as an argument.
  Return the pointer to trie node.

Explanation:

This function is in charge of setting the trie. It allocates memory and sets all the children to NULL. It also sets the code to the trienode code.

Deleting a Trie Node:
```
trie_node_delete(){
  for(i =0, i  < ALPHABET, i++){
    if(trie->children[i] != NULL){
      trie_node_delete(trie->children[i]);
    }
  }
  free(trie);
  return;
}
```

English Translation:
The function trie_node_delete takes in a trie node as its sole argument.
Free the argument.
This is a void function so just return;

Explanation:

This function is used to free all the memory that has been allocated to the trie. It only takes the trie node to free as its argument and returns void.

Creating a Trie (Initializing a Trie):

```
trie _create(){
  TrieNode *trie = trie_node_create(1);
  trie->children[0] = " ";
  Return trie;
}
```

English:
The trie_create function takes in no argument.
It creates a new trie by calling the trie_node_create function.
Sets the children at position 0 to a blank space.
Returns a pointer to a trie node.

Explanation:

This function is used to create a trie. It is used to create a trie and initiallize the root of the trie. This function calls the function trie_node_create, and by doing so is alloctating memory for the trie and initializing it.

Resetting a Trie:

```
trie_reset(){
  for (i = trie->code ; i < ALPHABET; i++)
        trie->children[i] = NULL;
   }
Return void;
}
```

English:
The function trie_reset takes in the TrieNode root of teh trie to reset.
This function setts all the nodes to NULL excluding the root.
The function returns void.

Explanation:

The purpose of this function is to reset the trie once it has been filled. This is important when compressing as it is more efficient to reset the trie once it has filled instead of just stopping when it is full. This results in a better compressed file. (The root stays the same);

Deleting a Trie:

```
trie_delete(){
  for (i = trie - 1 ->code ; i < ALPHABET; i++)
        trie->children[i] = NULL;
   }
   free(trie);
   return;
}
```

English:
The function trie_delete() takes in a root TrieNode as an argument.
Starting from the input node free everything after it.
Returns void.

Explanation:

This function is used to delete a portion of the main trie (a sub-trie, refer to end of DESIGN). This frees memory that was allocated to all nodes starting from the node that is passed through. This function may be used when the sub trie is no longer needed or space is needed but you cannot reset the whole node as you still need a portion of it.

Stepping Through the Trie:

```
trie_step(){
 for (i = trie - 1 ->code ; i < ALPHABET; i++)
        If (trie->children[i] == sym){
          Return trie;
         }
   }
```

```
}
```
English:

Explanation:

WORD TABLE - word.c

Create a Word:

```
word_create(){
  Word *wordtable = (Word *) calloc(1, sizeof(Word);
  wordtable->syms = syms;
  wordtable->len = (len *) calloc(len, sizeof(uint32_t);
  Return wordtable;
}
```

English:

The function word_create() takes in two arguments. The first one is a uint8 pointer syms and the second is a uint64 len.
Set the syms for wordtable equal to the syms that is passed in as an argument.
Set the len of the wordtable equal to that of len that is passed in.
Return the pointer to the word table.

Explanation:

This function takes in the parameters syms which is an array of symbols that a word represents. The function also takes the parameter len which is the length of the array of symbols. This function is important as it allocates memory and sets up and defines the word table.

Construct New Word With Given Symbol:

```
word_append _syms(){

}
```

English:

Expalnation:

Delete Word:

```
word_delete(){
  free(word->len);
  free(word);
  return;
}
```

English:

This function takes in a Word pointer as its argument.
Free the allocated memory for word->len
And then free the allocated memory to word
Return void.

Explanation:

This function is used to free the allocated memory for the word tables that have been created. The function takes in a word as its function and then frees both the memory allocated to word->len and the memory allocated to word. This function is important as it prevents memory leaks.

Create a Word Table:

```
wt_create(){
WordTable  *word = (uint16 *) calloc(uint16_max -1, sizeof(uint16);
Word[0] = word_create("", 0);
Return word;
}
```

English:

This function takes in no arguments.
Allocate memory of uint16_max - 1 for the word table.
In the first position of the array of word create a new word blank with a length of 0;
Return the pointer to the word table.

Explanation:

This function is used to create a word table. It is an array of words (or a pointer to a pointer). It sets up the root entry of the table by calling the function word_create, with a blank and a len of 0. The function returns a pointer to the word table.

Reset a Word Table:

```
wt_reset(){
for(int i = 1, i <= uint16_max - 1; i++);
   wordtable[i]->syms = NULL;
   wordtable[i]->len = NULL;
}
Return;
}
```

English:

The function wt_reset takes in a word table as its argument.
Iterate through the array of words and reset them to NULL or (0).
Reset the length of the word.
Return void.

Explanation:

This function resets the word table. This is used so that a file is decompressed properly.

Delete Word Table:

```
wt_delete(){
for (int i = 0, i <= UINT16_MAX  - 1, i++){
  word_delete(wordtable[i]);
}
return;
}
```

English:

This function takes in the word table as its argument.
Iterate through the array and call the function word_delete.
Return void.

Explanation:

This function takes in a word table as its argument. The purpose of this function is to free all memory that was allocated to a word table including that of the words. This function is made in order to avoid memory leaks when allocating memory for the table and its words.

I/O - io.c

Read Header:

```
read_header(int infile, FileHeader *header){
Int temp = read(infile, header, sizeof(header))
if (temp == -1)
  ERROR;
}
```

English:

The function takes in two arguments, a int file and a header.
Read the header into the fileheader.
If it equals a -1 then there was an error when reading.

Explanation:

This function takes in two arguments the int file and a header. The function's purpose is to get content from a file of a certain size and put it into fileheader. This is important as it will help check if the input magic numbers of the files match. It is also important for the permission.

Write Header:

```
Write_header(){
  Int temp = write(outfile, header, sizeof(FileHeader)

  if(temp == -1)

ERROR;


}
```

English:

Write_header function takes in two arguments, an outfile and a FileHeader pointer.
  Write the header into the outfile.
  If it is written incorrectly (temp == -1), then return an error.

Explanation:

The purpose of this function is to write the appropriate header to a file that is being compressed. When decompressing if this header that is being read matches the header that you set, then it can decompress. This function returns void, it is only meant to write to the outfile.

Reading the symbol:

```
read_sym(){
  if(buffer_index == max buffer or buffer_index == 0){
    read(infile, read_buffer, buff_limit);
    Buffer_index = 0;
  }
  for(0 to 8){
   Temporary = read_buffer & (1 << j) >>j;
   If(temporary == 1){
      Read_buffer = readbuffer[read_index] | (1 <<  j);
   }
}
  *sym = read_buffer[read_index];
  if(*sym == 0){
    Return false;
  }
}
```

English:
The function read_sym takes in 2 arguments, infile and a uint8_t *sym.
If the buffer is full or if it is empty read into the buffer from the infile.
Get the bits from lsb by shifting from the 0th position to the bitlen -1.
Put the value into the buffer by setting the bits at position buffer_index by oring it with the count.
If there is nothing left to read then return false.
If there is still things to read return true.

Explanation:

This function takes in the arguments infile and a uint8_t *sym. The purpose of this function is to read in symbols and essentially check if it has processed all the symbols in the file. It returns true if there are still symbols to be read and false if it is done

Buffer Pair:

```
buffer_pair(){
  if(buffer_index == buffer_limit)
    write(outfile, buffer, buffer_limit);
    Buffer_index = 0
  }
  for(0->bitlen){
     Temporary[i] = code & 1 << shift_by >> shiftby
     Shift_by ++;
  }
   while(shift_by - 1 > 0){
       IF(temporary[r] == 1){
           Buffer[buffer_index] = buffer[buffer_index] | (1<<shift_by);
        }
    }

  //repeat for sym.

  for(0->bitlen){
     Temporary[i] = sym & 1 << shift_by >> shiftby
     Shift_by ++;
  }
   while(shift_by - 1 > 0){
       IF(temporary[r] == 1){
           Buffer[buffer_index] = buffer[buffer_index] | (1<<shift_by);
        }
    }
}
```

English:

Function buffer_pairs takes in four arguments, outfile, code, sym, and bitlen.
  If the buffer is full or if it is empty write the buffer out to outfile.
  Use a for loop to get the individual bits of code from lsb (convert to binary).
  Use a while loop to store the bits into a position of an array.
  Use a for loop to get the individual bits of sym from lsb (convert to binary).
  Use a while loop to store the bits into a position of an array.
  Returns void.

Explanation:

This function takes in four arguments, an outfile, where the buffer is to be written if it is full, a code that you want to buffer, a sym that you want to buffer, and a bitlen for the code. This function is used to compress a file as all the codes and symbols in the buffer are placed from lsb to msb.

Flush Pairs:

```
flush_pairs(){
  write(outfile, buffer, buffer_index);
}
```

English:

The function flush_pairs takes in the outfile as its only argument.

  Write the buffer to the outfile (write buffer index amount of bits);

Explanation:

This function has only the argument of outfile. This function is used to write the buffer pairs into the outfile if it is less than the max size of the buffer. This function is used only when there are no more bytes to read from the file and the buffer is not full.

Read Pair:

```
read_pair(){
   if(index == 0 or index == Max){
       read(infile, buffer, Max);
    }
    for(i = 0; 0->bitlen){
       Binary = buffer_2[index] & (1 << shift_by) >> shift_by;
       if(binary == 1){
          Decimal value = decimal value + pow(2, i);
       }
    }
    Code = decimal value;

        for(i = 0; 0->8){
       Binary = buffer_2[index] & (1 << shift_by) >> shift_by;
       if(binary == 1){
          Decimal value = decimal value + pow(2, i);
       }
    }
    sym = decimal value;
    if(code == stop_code){
       Return false;
    }
     Return true;
}
```

English:

The function takes in four arguments, infile, code pointer, sym pointer, and bitlen

```
  If the buffer is empty or if it is full read in from infile (Max amount)
  Use a for loop to see what bits are set in the first code of the pair.
  Hold the value of the bit in a temporary.
  If the temporary is 1 then get 2 to the power of the count and add it to the previous value.
  Set the buffer at the index = to the temporary.
  Set the code pointer to the buffer at the index.

  Use a for loop to see what bits are set in the first sym of the pair.
  Hold the value of the bit in a temporary.
  If the temporary is 1 then get 2 to the power of the count and add it to the previous value.
  Set the buffer at the index = to the temporary.
  Set the sym pointer to the buffer at the index.

  If the code pointer is equal to the stop code return false, otherwise return true.
```

Explanation:

This function takes in four arguments, a infile, a pointer to code, a pointer to sym, and a bitlen. This function is used to read pairs from the compressed file. It is the opposite of buffer pairs. This function returns true if there are still pairs to read and false if there are no more pairs to read.

Buffer Word:

```
buffer_word(){
 for(i = 0; i to w->len){
   for(j = 0; j < 8){
      Temporary = ((w->syms[i] & 1 << j >>j)
       if(temporary == 1){
          Buffer[index] = buffer[index] | 1<< j
       }
    }
    if(index == limit){
       write(outfile, buffer, buffer_limit);
       Index = 0;
    }
 }
}
```

English:

```
This function takes in two arguments, outfile and word.
  Use a for loop to go through all the characters of the word.
      Use a for loop to get all the bits of the current character and put them into a buffer at current index.
   If the buffer index is equal to max then write the buffer to the outfile.
Return void.
```

Explanation:

This function takes in the arguments outfile and word. The function is used put a word from the trie and put it into a buffer. If the buffer is full the function writes the buffer out to the outfile. The function returns void.

Flush Words:

```
flush_words(){
    write(outfile, buffer, buffer_index);
     return;
}
```

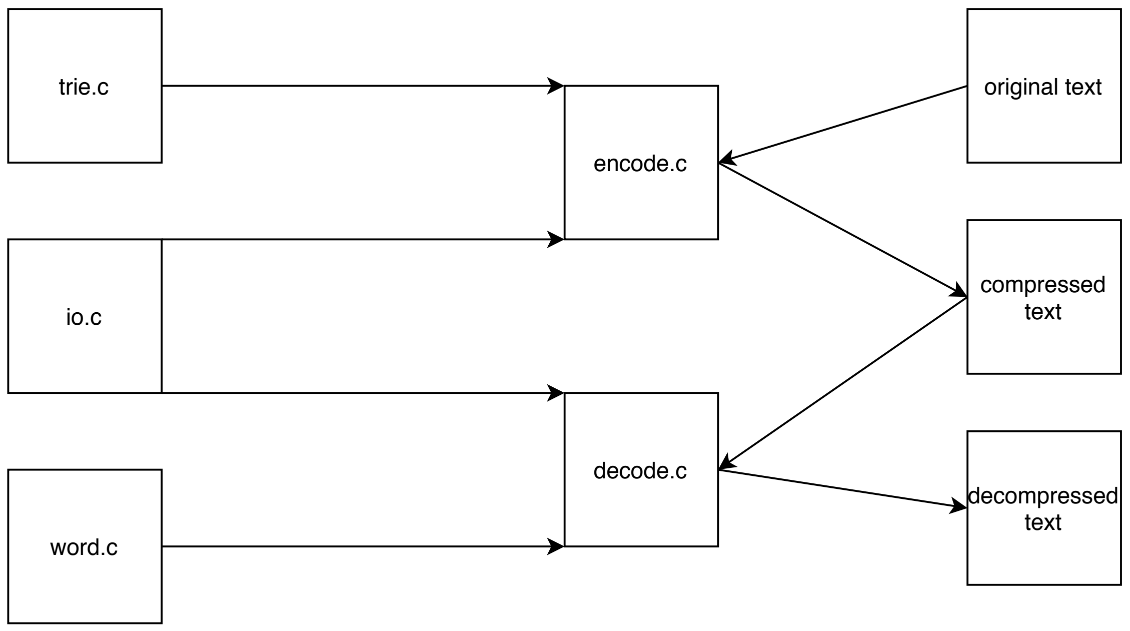
English:

```
The function flush_words takes in an outfile as its argument.
    Write the buffer to the outfile (write out buffer_index amount of bytes into the outfile).
     Return void.
```

Explanation:

This function uses the outfile as its only argument. This function is used to write the word into the file that is being outputted. It is only used to write the end of the file or when the buffer does not fill by the time all the words have been read.

General Flow Chart



This is a general overview of file connections.
Look at psudoecode and explanation for individual function purposes

Notes:
1. Flags:
  -i input, specify the input (standard input is default.)
  -o output, specify the output (standard output is default)
  -s prints out the statistics of compression/ decompression
2. When compressing buffer from least significant bit.
3. Make sure to check endian.