Purpose of the lab:

The purpose of the lab is to build a program using bloom filter, hash table, and linkedlist to check texts to see if there are any bad words. The program will print out all the bad words and the new versions of the words(if those bad words appear in newspeak.txt).

Command options:

-h: print out help message and program information
-t: specifies hash table size
-f: specifies bloom filter size
-m: enable move to front rull
-s: print statistics to stdout

Files:

Banhammer.c: main function to run the program
Message.h: provided texts for bad speak, goodspeak, and mixspeak
Speck.c and .h: provided functions to use like hash function(multiply the salt to get index for bloom filter)
Parser.c and .h: regex parsing module
Bv.c and .h: bit vector interface
Bf.c and .h: bloom filter functions
Ll.h and .c: linked list functions
Ht.c and .h: hash table functions

Overall/general structure of the program:

1. Use hash function in speck.c to multiply salts to get 3 indexes for each word and insert it to bloom filter
2. When the program reads through each word, check if each word indexes in bloom filter == 1? If no, meaning the whole program didn't store this word, meaning this is a good word.
3. If bloom filter indexes for the word == 1, then go search in hash table
4. We store bad words in the hash table, if any hash table index conflicts, then we create a link list in that index to save the second bad words.
5. Move to front: if we find a word in the linked list, we can move that word to the front of the linked list(meaning the frequency of this word is high), this will speed up the search speed if this word appears again next time.
6. In the main function, we should use define WORD "([a-zA-Z0-9](-|')?)+" and use regex in parser.h to limit user input.
7. Bv and bf are the helper files to set up bloom filter

Pseudocode:
Bf.c:
Bf_create: check pdf
Bf delete: bv_delete first, then free pointer and set it to NULL
Bf_size: return bv_length

Bf_insert: bv_set_bit(bf->filter, hash(salt 1, oldspeak) % bv_length)
bv_set_bit(bf->filter, hash(salt 2, oldspeak) % bv_length)
bv_set_bit(bf->filter, hash(salt 3, oldspeak) % bv_length)
bf_probe:
a = hash(salt 1, oldspeak)
b = hash(salt 2, oldspeak)
c = hash(salt 2,3, oldspeak)
l = bv_length(bf->filter)
if 3 bits are set, then return true
Bf_count:
For loop until linked list size, if bv_get_bit ==1, then count++

Node.c
Node_create
If oldspeak / newspeak == null, then set both to null, otherwise strdup
Set next and prev to null

Node_delete: free oldspeak and newspeak, free pointer

Ll.c
Ll Create:
Head and tail = node_create(“”, “”)
Head next = tail
Tail prev = head
Mtf = mtf
Ll delete: node_delete first, and fre pointer
Ll lookup: for loop loop until tail, strcmp if word matched v->oldspeak
For mtf:
v->prev->next = v->next
v->next->prev= v->prev
v->next = ll->head->next
v->prev = ll->head
ll->head->next->prev = v
ll->head->next = v
Ll insert:
If node == null, then create node
Node *n = node_create(oldspeak, newspeak)
n->prev = ll->head
n->next = ll->head->next
ll->head->next->prev = n
ll->head->next = n
ll->length += 1

Ht.c:
Ht_lookup:
Return ll_lookup(ht->list[hash(salt, oldspeak), ht_size(ht), oldspeak]
Ht_insert:
If !ht->lists[(ht_idx % size)]
Ll create then ll insert
Else ll insert

Ht_count: loop for the size, and count++

Banhammer.c:

```
Command line
Create bloom filter
Fopen bad speak and newspeak, then bf_insert and ht_insert
Regex stuffs
If bf_probe == true, check bf
    If ht_lookup ==null, check ht
        Cheek old speak or new speak
            Ll_insert oldspeak or new speak

Print message for oldspeak, newspeak, and both
Print statics
Free memory
```