Assignment 6:

This assignment is related to the George Orwell novel *1984*, a famous dystopian novel in which the thoughts, words and actions of the citizens are controlled by a totalitarian government. In order to replicate this through a program, we implement linked lists, hash tables and bloom filters to first make a record of certain badspeak and oldspeak words that should not be used by people before parsing through the user input to determine whether the words used by the user can get them accused of thoughtcrime. In the case that they use words without a newspeak translation— badspeak— they are accused of thoughtcrime and sent to joycamp whereas if they simply use oldspeak words that have newspeak translations, they are sent to counsel.

Pre-Lab Part 1: pseudocode for inserting + deleting elements from Bloom filter:

```
void bf_delete(BloomFilter **bf):
    free((*bf)->filter)
    free(*bf)
    *bf = NULL

void bf_insert(BloomFilter *bf, char *oldspeak):
    hash(primarySalt, oldspeak)
    hash(secondarySalt, oldspeak)
    hash(tertiarySalt, oldspeak)
```

Pre-Lab Part 2: pseudocode for functions in linked list ADT

```
LinkedList *ll_create(bool mtf):
    LinkedList *l = (LinkedList *) malloc(sizeof(LinkedList));
    l →  length = 0;
    l → mtf = mtf
    l →  head = node_create(NULL, NULL)
    l → tail = node_create(NULL, NULL)
    l →  head → next = l → tail
    l →  tail → prey = l → head
    return l

void ll_delete(LinkedList **ll):
    while (*ll)-> head = null
        node *temp = (*ll)->head->next
        node_delete(&(*ll)->head)
        (ll)->head = temp
    free(*ll)
    *ll = NULL

uint32_t ll_length(LinkedList *ll):
    return l->length

Node *ll_lookup(LinkedList *ll, char *oldspeak):
    for (Node *n = l → head → next; n != l → tail; n = n→ next)
        if compare n→ oldspeak & oldspeak
            if(l → mtf)
                x->prev->next = x->next
                x->next->prev = x->prev
                x->next = l->head->next
                x->prev = l->head
                l->head->next->prev = x
                l->head->next = x
            return n
        return NULL

void ll_insert(LinkedList *ll, char *oldspeak, char *newspeak):
    Node *x = node_create(oldspeak, newspeak)
    x → next = l →  head → next
    x → prey = l → head
    l → head → next → prey = x
    l → head → next = x
    l → length += 1

void ll_print(LinkedList *ll):
    for (Node *n = l → head → next; n != l → tail; n = n→ next)
        print(n)
```

**Pre-Lab Part 3*: regular expressions to match words with

[a-zA-Z0-9_]+((-|’)[a-zA-Z0-9_]+)* [referred to Sahiti]

used to parse stdin

Pseudocode:

node.c: [referred to Sahiti’s code]

```
create:
    malloc node
        if malloc failed: return null
    if oldspeak != null
        strdup oldspeak to n->oldspeak
    else
        n->oldspeak == null
    if newspeak != null
        strdup newspeak to n->newspeak
    else
        n->newspeak == null
    n->next =  null
    n->prev = null

delete:
    free ((*n)->oldspeak)
    free((*n)->newspeak)
    free(*n)
    set *n = null
print:
    if n->oldspeak & n->newspeak != null
        print both
    if n->oldspeak != null & n->newspeak == null
        print n->oldspeak
```

ll.c: [referred to Eugene and Sahiti]

```
*refer to lab part 2*
```

hash.c: [referenced Eugene’s code]

```
create:
    given in asgn doc
delete:
    iterate from i =0 to i<ht_size(*ht)
        if(*ht)->lists[i]
            ll_delete(&(*ht)->lists[i])
    free((*ht)->lists)
    free(*ht)
    *ht  = null
size:
    return size
```

```
lookup:
    linkedlist *s = ht->lists(hash(salt, oldspeak)%ht_size)
    if malloc for s failed
        return null
    node *lookup_node = ll_lookup(s, oldspeak)
    return lookup_node
insert:
    hashKey = hash(salt, oldspeak)%ht_size
    if(ht->list[hashKey] == null)
        ht->lists[hashKey] = ll_create(ht->mtf)
    ll_insert(ht->lists[hashKey], oldspeak, newspeak);
print:
    from i=0 to ht_size
        linkedlist *p = ht->lists[i]
    if p!= null
        ll_print(p)
```

bv.c:

```
same from previous assignment
edit to fit size of ht
```

bf.c:

```
create:
    given in asgn doc
delete:
    bv_delete(&(*bf)->filter)
    free(*bf)
    *bf = null
length:
    return bv_length(bf->filter)
insert:
    hash1 = hash(primarySalt, oldspeak)%bf_length
        set bit
    hash2 = hash(secondarySalt, oldspeak)%bf_length
        set bit
    hash3 = hash(tertiarySalt, oldspeak)%bf_length
        set bit
probe:
    firstBit = get_bit(bf->filter, primaryHashVal)
    secondBit = get_bit(bf->filter, secondaryHashVal)
    thirdBit = get_bit(bf->filter, tertiaryHashVal)
    if(firstBit & secondBit & thirdBit == 1)
```

```
        return true
    else
        false
```

banhammer.c:

1) loop through getopt
2) initialize/create bloom filter and hash table
3) read in badspeak words from badspeak.txt (badspeak has no translation)
    - for each badspeak word
        - insert into bloom filter
        - insert into hash table -> ht_insert(ht, “word”, null)
4) read in newspeak words from newspeak.txt (fscanf(%s %s\n)
    - for each olspeak, newspeak pair
        - insert oldspeak into bf
        - insert (oldspeak, newspeak) into ht -> ht_insert(ht, “word”, “translation”)
5) use regex to parse stdin
    - for each word, ask if its in bf
    - if yes, ask ht
    - if word is in bf & ht
        - does word->newspeak != null?
            - if not, user used badspeak -> joycamp
            - If yes, counsel user