Pre-Lab Part 1

1.Pseudocode for Bloom Filter:

```
Bf_delete(key)
For i in bf:
        If theres a 1 in bv: then clear it
free(bf)

bf_insert():
Int a=hash(key)
Int b=hash2(key)
Int c=hash3(key)
If bv(hash)  does ! = 1:
        bv_insert(hash(a))
        bv_insert(hash(b))
        bv_insert(hash(c))

bf_probe(key):
*Do this all for the hash salts
If bv(hash(keys)) == 1:
        Return true
Return false
```

2.A bloom filter with m bits and k hash function will take O(k) time because it directly checks the bit vector array at that index and returns 1 or 0. It checks each hash function spots.The space complexity will be O(m), it increases based on bit size.

Pre-Lab Part 2

1.

2.Pseudocde for Linked List

```
Ll_create(data):
Allocate space for list array
Put data inside newly created node
Set pointer to next to Null

Ll_node delete(*n):
free(current listNode) at *n
*n=NULL

Ll_delete:
Create listnode *temp=*head
While (temp !=NULL):
        free(temp)
        temp=temp->next
listnode=NULL

ll_insert():
If Linked node with data doesn't already exist:
        Create new Listnode =(call ll_create function)
        listnode->next=current head
        *head=new list_node
        Return listnode
Else:
        Ll_lookup and get node with same data and replace data inside

ll_lookup(**head, *key)):
while(*head !=NULL):
        if value at head node equals to the key you want to find:
                Return the head pointer to node
        Else:
Return NULL

DESIGN:
main()
//must input shfm or b
File = *stdin
Read In baspeak.txt
Read in Goodspeak.txt
Compile Regular expression with parser to take in words
DEFAULT is Move_to_front is false
```

while(getopt(argc,argv,”sh:f:mb”)!=-1) :
    switch(opt)
    Case s:
        Don’t print letter, and prints statistic
    Case h:
        Hash table size
    Case f:
        Bloom Filter size
    Case m:
        Indicate move-to-front
    Case b:
        Not use move-to-front

Create *BloomFilter and *Hashtable with respective size

While file !=NULL :
    Read in forbidden words and check if it’s a word with Regex
    Insert word in bloom filter
    GoodSpeak(oldspeak=forbidden word,newspeak=NULL)
    Insert goodpeak struct into Hashtable

While file != NULL:
    Read in pairs of oldspeak,newspeak
    ht_lookup(oldspeak)
    If oldspeak in hashtable :
    Goodspeak *gs=(oldspeak=oldspeak,newspeak=newspeak)
    ht_insert(gs)
Read in words from stdin and then check if regex; until EOF
Check if each word is a forbidden word or has translation
If the word in Hash table has not newspeak word, then store in joycamp array
If word in Hash table has translation then store in translation array

If Case !S:
    If forbidden words, printf joycamp message
    If replaceable wods, print message
    If it is clean, then print a clean message.
If Case S:
    Print statistics of loads.

Goodspeak struct(oldspeak,newspeak):
    *gs=allocate memory

gs->oldspeak=oldspeak
gs->newspeak=newspeak

HashTable Function():

Ht_create():
Allocate memory for *ht
Build hash Salt;
Allocate array for storing pointers to head

Void Ht_delete():
For i in range(ht->length):
If hashTable[index] !=NULL:
Free data inside hashtable at index i
Hash table at index = NULL

Free whole hash table

Ht_Insert(*head,goodspeak):
Hash(key)
If hash table empty at that index then call LL_create
(create new node to put inside table)
If not empty then call LL_insert
(links on to existing node in table)

Ht_Lookup(*head,key):
Hash(key)
Call LL_lookup(*head,key)
(checks if pointer of listnode at key exists in Hashtable)
Returns pointer to hash table that contains that word

Ht_Print():
Prints entire hash table pointers