<u>Pre-Lab Part 1</u>

1.Pseudocode for Bloom Filter:

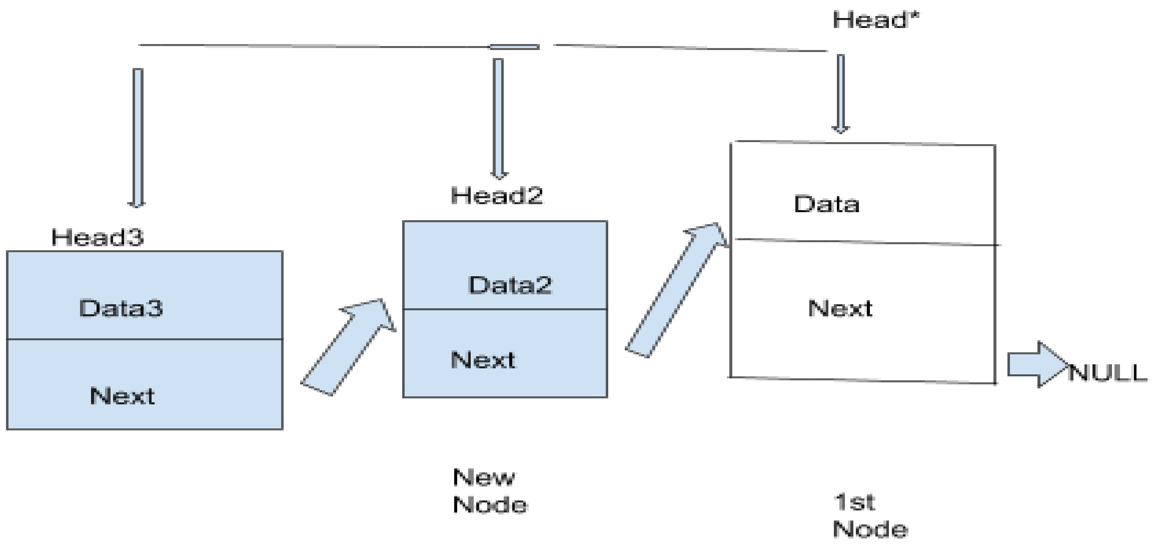
```
Bf_delete(key)
For i in bf:
        hash(key)
        bv_clear_bit(hash(key)))
free(bf)

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):
If bv(hash(key)) == 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.

<u>Pre-Lab Part 2</u>



1.

2.Pseudocde for Linked List

```
Ll_create:
Allocate space for list array
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():
Create new Listnode =(call ll_create function)
Store data into listnode (temp->data)
listnode->next=current head
*head=new list_node
Return listnode

ll_lookup(**head, *key)):
while(*head !=NULL):
    If strcmp(head->data,key) equals to one another:
        Return *head
    Else:
        *head= listnode->next

Return NULL

DESIGN:
main()
//must input shfm or b
File = *stdin
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 (fscanf) 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 (fscanf)
    ht_lookup(oldspeak)
    If oldspeak in hashtable :
    Goodspeak *gs=(oldspeak=oldspeak,newspeak=newspeak)
    ht_insert(gs)

If Case ~S:
    If forbidden words, printf joycamp message
    If replaceable wods, print message
    If it is clean, then print a clean message.


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)
    Call LL_insert(*head,goodspeak key)
    (inserts pointer of linked list node to Hashtable)

Ht_Lookup(*head,key):
    Hash(key)
    Call LL_lookup(*head,key)
    (checks if pointer of listnode at key exists in Hashtable)

Ht_Print():
    Prints entire hash table pointers
```