# Assignment 6: Design

Down the Rabbit Hole and Through the Looking Glass: Bloom Filters, Hashing, and the Red Queen's Decrees

***Pre-Lab Questions***

*Write down the pseudocode for inserting and deleting elements from a Bloom filter.*

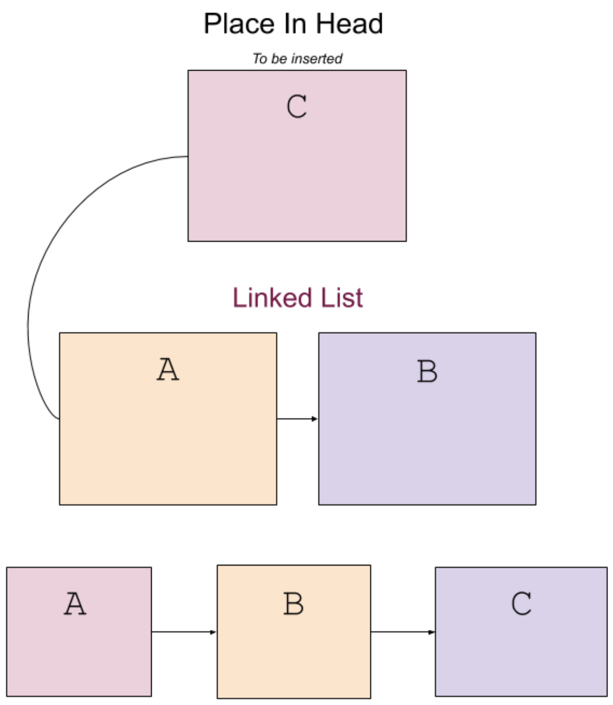
Assuming we have access to the bit vector struct we created in previous labs the pseudo-code is deceptively simple.

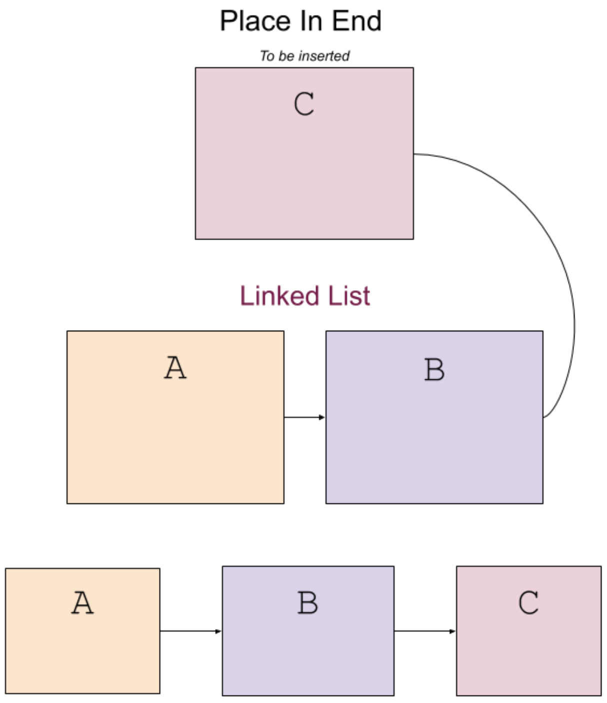
```
insert(filter, key):
    bv_set_bit(filter, hash(salt1, key) % bf->size)
    bv_set_bit(filter, hash(salt2, key) % bf->size)
    bv_set_bit(filter, hash(salt3, key) % bf->size)

delete(filter, key):
    bv_clr_bit(filter, hash(salt1, key) % bf->size)
    bv_clr_bit(filter, hash(salt2, key) % bf->size)
    bv_clr_bit(filter, hash(salt3, key) % bf->size)
```

*Assuming that you are creating a bloom filter with m bits and k hash functions, discuss its time and space complexity.*

The time complexity of this bloom filter would be O(k), or essentially however long it takes to get your hash. The space complexity would be O(n) , n being to avoid false positives.

*Draw a picture to show the how elements are being inserted in different ways in the Linked list.*





*Write down the pseudocode for the above functions in the Linked List data type.*

```
ListNode *ll_insert ( ListNode **head , HatterSpeak *gs)
        ListNode *A = allocate
        A->gs = gs
        A->next = (*head)
        (*head) = A
        return A


void ll_delete ( ListNode *head )
        ListNode *n = head
        ListNode *save
        while(n!=NULL)
                save = n->next
                free(n->gs)
                free(n)
               n = save;



ListNode *ll_lookup ( ListNode **head , char *key )
        ListNode *n = (*head)
        ListNode *prev = NULL
        while(n!=NULL)
                if (strcmp(n->gs->oldspeak,key)==0)

                    if(move_to_front)
                        if(prev != NULL)
                            prev->next = n->next
                            n->next = (*head)
                            (*head) = n


                        return n
                else

                        prev = n
                        n = n->next


                return NULL

ListNode *ll_node_create ( HatterSpeak *gs)

        ListNode *l = alloc
        l->gs=gs
        l->next = NULL
        return l
}
```

# Pseudo-Code

***ONLY SHOWING FILES NOT SHOWN IN PREVIOUS LAB OR IN PRELAB***

## *hatterspeak.c*

```
Include all required Headers


Move_to_front = false
Stats = false
File = oldspeak.txt
File2 = hatterspeak.txt
Int hashSize
Int bloomSize
while(inputs on command line)
        Switch
                Case h
                        hashSize = input
                Case f
                        bloomSize = input
                Case m
                        Move_to_front = true
                Case b
                        Move_to_front = false


Bloomfilter *bf
Bloomfilter *ht
// Process Oldspeak
Regex expr
Int y
while(next_word!=null)
        Char word1
        Ptr[y] = allocate
        Ptr[y]->oldspeak = word1
        ptr[y]->hatterspeak = NULL
        bf_insert(ht,ptr[y])
        ht_insert(ht,ptr[y])
        y+=1

// Process hatterspeak
Regex expr
Int y
while(next_word!=null)
        Char word1
        Char word2
        Ptr[y] = allocate
        Ptr[y]->oldspeak = word1
        ptr[y]->hatterspeak = word2
        bf_insert(ht,ptr[y])
        ht_insert(ht,ptr[y])
        y+=1
// process stdin
Regex expr
List violations
List translations
while(next_word!=null)
        lowercase(nextword)
        Char looking
        Looking = next_word
        if(bf_probe(bf,looking))
                if(result=ht_lookup(ht,looking))
                        translations[g]=result->oldspeak
                        Translation[g+1] = result->hatterspeak
        Else
                Violations = result->oldspeak

if(not stats)
        print("msg to user)
        print("violations")
        print(translations)
if(stats)
        Print stats
```

*Array.c*

```
void createList(list *a, size_t size)
  a->list = alloc
  a->used = 0
  a->size = size
void insertList(list *a, char* element)
    if (a->used == a->size)
           a->size += 2
           a->list = realloc
  a->list[a->used++] = element
```

*INFLUENCE FROM STACK IMPLEMENTATION OF PREVIOUS LAB AND TUTORIAL FOR HOW TO CREATE DYNAMIC INT ARRAYS*

*(https://stackoverflow.com/questions/3536153/c-dynamically-growing-array)*

*Used to store words after they are found to be missspeach*

# Hash.c

*Only showing what we had to implement*

```
void ht_delete ( HashTable *ht){
    for(unsigned int i = 0 ; i<ht->length; i++)
        ll_delete(ht->heads[i]
        free(ht -> heads)
        free(ht)
uint32_t ht_count(HashTable *h)
        int x = 0
        for(unsigned int i=0; i<h->length; i++)
                if(h->heads[i] != NULL)
                        x+=1
        return x



ListNode *ht_lookup ( HashTable *ht , char * key )
        seeks+=1
        uint32_t index = hash(ht->salt, key) % ht->length
        void *x = ll_lookup(&ht->heads[index], key)
        if(x!=NULL)
                return x
        else
                return NULL;

void ht_insert ( HashTable *ht , HatterSpeak *gs)


        seeks-=1
        links-=1
        if(ht_lookup(ht, gs->oldspeak)==NULL)
                uint32_t index = hash(ht->salt, gs->oldspeak) % ht->length;
                if(ht->heads[index] != NULL)
                        ll_insert(&ht->heads[index], gs);
                else
```

```
ListNode *node = ll_node_create(gs);

ht->heads[index] = node;
```