ASGN 6 DESIGN

BLOOM FILTERS

- COULD PRODUCE FALSE POSITIVES

- TAKE EVERY POTENTIALLY NONSENSE WORD AND HASH INTO A BLOOM FILTER.

- 3 SALTS (ALTERATIONS OF HASH) TO ENSURE INCLUSION
  - ALL 3 MUST BE SET TO CONFIRM WORD IS PRESENT

- IF BLOOM FILTER REJECTS ALL WORDS PERSON IS INNOCENT OF OLDSPEAK
  - IF ANY WORD IS PRESENT THEY ARE GUILTY

- CONSULT HASH TABLE - IF WORD IS THERE AS NONSENSE THEN THEY GO TO DUNGEON

- IF WORD IS NOT FORBIDDEN, AND PASSED BOTH FILTERS, HASH TABLE WILL PROVIDE TRANSLATION FROM NONSENSE → APPROVED

CASES:

- APPROVED WORDS WILL NOT APPEAR IN BF
- WORDS THAT SHOULD BE REPLACED, WHICH WILL HAVE A MAPPING FROM THE OLD WORD, OLDSPEAK, TO THE NEW WORD, HATTERSPEAK
- WORDS WITHOUT TRANSLATIONS TO NEW APPROVED WORDS MEANS OFF TO DUNGEON

PRE-LAB 1

1) WRITE DOWN PSUEDOCODE FOR INSERTING AND DELETING ELEMENTS FROM A BLOOMFILTER.

PSUEDOCODE FOR ENTIRE BLOOM FILTER ON NEXT PAGE
USES bv.h + bv.c FROM PREVIOUS LAB

2) ASSUMING YOU ARE CREATING A BLOOMFILTER WITH m BITS AND K HASH FUNCTIONS, DISCUSS ITS TIME AND SPACE COMPLEXITY.

THE SPACE COMPLEXITY OF A BLOOMFILTER WILL BE DIRECTLY RELATED TO THE NUMBER OF BITS, SO O(m). BLOOMFILTERS ALSO HAVE VERY EFFICIENT SEARCH AND INSERT, RELATED TO THE NUMBER OF HASH FUNCTIONS K, SO O(K).

BLOOM FILTER PSUEDO CODE:

```
typedef struct BloomFilter {
    - defined in header file
    - salts are arrays of size 2 bc they need to be 128 bits
    - filter is the Bitvector
}

bf_create(uint32_t size):
    - given in lab document
    - initialize salts
    - create bit vector + malloc BloomFilter

bf_delete(BloomFilter *bf):
    - bv_delete(bf→filter)          (maybe free bf→filter)
    - free(bf)

bf_insert(BloomFilter *bf, char *key):
    - hash(key)
    - use salts with hash to produce 3 indices

    - bv_set_bit() @ each index (modulo size)

bf_probe(BloomFilter *bf, char *key):
    - hash(key)
    - use salts with hash to produce 3 indices

    - if(bv_get_bit() for all 3 salts): (modulo size)
        - return true
    - else : return false
```

HASH TABLES

- "ENTRIES" IN HASH TABLE ARE OF TYPE HatterSpeak
  - oldspeak + hatterspeak strings

- IF A WORD IS IN BLOOM FILTER – EITHER NONSENSE OR NEEDS TO BE TRANSLATED
  - HASH TABLE LOCATES WORD (AS KEY) AND PROVIDES TRANSLATION
  - WORDS WITHOUT TRANSLATION = DUNGEON

- HASH USING PROVIDED SPECK CIPHER/ HASH FUNCTION

LINKED LISTS

- RESOLVES HASH COLLISIONS

- EACH NODE OF THE LINKED LIST CONTAINS A GOODSPEAK STRUCT
  - CONTAINS OLDSPEAK + HATTERSPEAK TRANSLATION IF EXISTS
  - OLD SPEAK WORD IS USED AS KEY

TWO IMPLEMENTATIONS

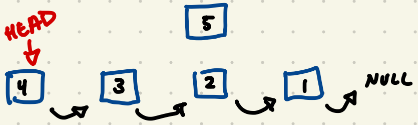
- INSERTING EACH NEW WORD @ FRONT OF LIST
- INSERTING EACH NEW WORD @ FRONT <u>BUT</u> EACH TIME IT IS SEARCHED FOR IT IS MOVED TO THE FRONT OF THE LIST.

- KEEP TRACK OF AVERAGE # OF LINKS FOLLOWED
  - IN LOOKUP

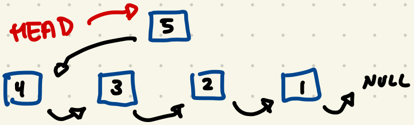
PRE LAB #2

1) DRAW PICTURES TO SHOW HOW ELEMENTS ARE BEING INSERTED IN DIFFERENT WAYS

BEFORE:



AFTER:



2) WRITE DOWN THE PSVEDOCODE FOR THE LL FUNCTIONS

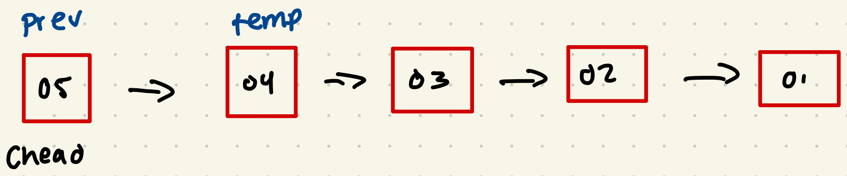
CAN BE FOUND BELOW

LINKED LIST PSUEDO CODE

```
struct ListNode {
  - HatterSpeak *gs
  - ListNode *next
}

ll_node_create (HatterSpeak *gs) :

    - n = malloc (sizeof(ListNode)) ( n is name for ptr to node)
          - CHECK IF malloc WORKED
    - n-> gs = gs
    - n -> next = NULL   (CURRENTLY LAST NODE)
    - RETURN - n (ptr to newly created node)

ll_node_delete (ListNode *n) :

    - free(n -> gs -> oldspeak)    [isn't freed anywhere else]
    - free(n-> gs -> hatterspeak)
    - free (n)

ll_delete (ListNode *head) :

    - temp = head  (might need to be copy instead of another reference?

    - while (temp-> next) :  (AKA temp-> next != NULL /NOT LAST NODE)
        - ll_node_delete (head??)
        - temp = temp -> next          MAYBE SWAP IF ERROR w/ACCESS
        - MAY NEED TO ALTER HEAD POINTER IN SOME FORM

* ll_insert (ListNode ** head, HatterSpeak *gs) :
    - new = ll_node_create (gs
                                        CHECK IF NODE EXISTS
    - new -> next = *head  ( point your new node to the old head)
                           ( next ptr now = ptr to previous head)

    -*head = new  ( move head pointer to new node)

    - possibly return *head
    - NEED DOUBLE POINTER SO CHANGE PERSISTS OUTSIDE OF FUNCTION

* ll_lookup (ListNode **head, char *key):
    - temp = head*
    - while (temp-> next):
        - if temp -> gs -> oldspe = Key  (strcmp) (IF MOVE TO FRONT]
            - return temp / set head to temp (multi step)
    -if never found return NULL
```

HASH TABLE PSUEDOCODE

```
struct  HashTable  {
    uint 64   salt[2];
    uint 32   length;
    ListNode  ** heads;
}

HashTable * ht_create(uint32 length:
    - given in lab document

void  ht_delete (HashTable *ht)
    - ll_delete (heads) (THIS TAKES CARE OF DELETING HS
    - free (heads) (FREE ARRAY ITSELF)
    - free (ht)
    - DO YOU HAVE TO FREE SALT?

uint 32_t ht_count (HashTable *h):
    - make linked list get-len function
    - loop through heads[] (ARRAY OF * TO LL)
        - IF NOT NULL, ADD LENGTH OF EACH TO COUNTER.

ListNode *ht_lookup (HashTable *ht, char *key):
    - hash key to find index of hash table it should
      be inserted in
    - call ll_lookup() ON THAT LINKED LIST
        - RETURN ITS RESULT
        - WILL BE NULL IF NOT THERE

void  ht_insert (HashTable *ht, Hatterspeak *gs):
    - gs->oldspeak is key so hash gs->oldspeak for
      hash table index

    · @  index call ll_insert (gs) to create with
      correct hatter speak object.
```



LEXICAL ANALYSIS W/ REGEX

- NEED FUNCTION TO PICK WORDS FROM AN INPUT STREAM
  - VALID WORDS CAN INCLUDE CONTRACTIONS
    - ACCOUNT FOR HYPHONS, APOSTROPHES, AND UNDERSCORE (REGEX ✓)

- WRITE REGEX OURSELVES

- next_word() :
  - COMPILED REGEX EXPRESSION = USE regcomp() BEFORE PASSING

- REMEMBER TO USE clear_words() TO FREE MEMORY

- TRANSFORM WORDS TO LOWERCASE BEFORE PASSING TO BF + HT

ACCESSING FILES

- OPEN FILES W/ FILE *name = fopen(oldspeak.txt);

- FORBIDDEN WORD = hatter_create(old, NULL)

- CREATE BUFFER → STORE OLDSPEAK/HATTERSPEAK PAIRS
  - HATTERSPEAK.txt FILE
  - CREATE A CORRESPONDING HATTER STRUCT FOR EACH PAIR

- HASH INDEX IS OLDSPEAK

- PASS THROUGH BLOOM FILTER BUT NO TRANSLATION = FORBIDDEN
  - NONTALK

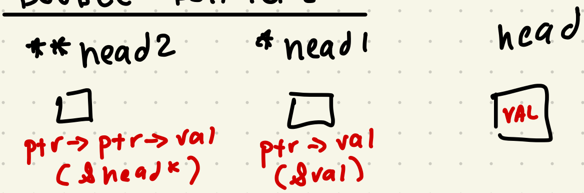
- 3 POSSIBILITIES : ONLY NONTALK (FORBIDDEN), ONLY OLDSPEAK, SOME OF BOTH

MAIN PSUEDOCODE

```
- get opts:
· initialize bloom filter and hashtable
- PARSE OLDSPEAK.TXT (AKA FORBIDDEN WORDS)
    - CREATE BUFFER (STRING / char[])
    - INSERT INTO BLOOM FILTER
    - CREATE HATTER SPEAK OBJECT (HS=NULL)
    - INSERT INTO HASH TABLE
    - CLOSE FILE

- PARSE HATTERSPEAK.TXT (AKA OLDSPEAK WORDS WITH TRANSLATION)
    - CREATE BUFFER (STRING / char[])
    - CREATE HATTERSPEAK STRUCT
    - PASS OLD SPEAK INTO BLOOM FILTER
    - PASS ENTIRE STRUCT INTO HASH TABLE
    - CLOSE FILE

- USER INPUT AND OUTPUT / REGEX
  - INITIALIZE REGEX OPERATION
      - DO CHECKS
  - CREATE LINKED LIST OF FORBIDDEN WORDS
  - CREATE LINKED LIST OF TRANSLATABLE WORDS
  WHILE (LOOP THROUGH WORDS OF USER INPUT W/ REGEX):
      · SET LL NODE OF WORD
      IF NODE WAS CREATED PROPERLY
          SAVE OLDSPEAK                          convert to lowercase
          IF HATTERSPEAK OBJECT EXISTS
              - CREATE STRUCT & ADD TO LL OF TRANSLATABLE
          ELSE
              - CREATE STRUCT & ADD TO LL OF FORBIDDEN
  PRINT OUTPUTS ACCORDINGLY
```

EXTRA NOTES

DOUBLE POINTERS:



head = 10 changes val -> 10

*head1 = 20 changes val -> 20

**head2 = 30 changes val -> 30

head1 ... changes what head1 points 2

*head2 ... changes val of head1/ what it points to

- NEED DOUBLE POINTER TO CHANGE PTR OF FORMAL PARAMETER

SOURCES

- ADVICE FROM OLY'S AND MAXWELL'S LAB SECTIONS
- VARIOUS PIAZZA POSTS
- "HOW TO LOWERCASE A STRING" STACK OVERFLOW