Adrian Murillo

Prof Darrel Long

CSE 13S

November 17th, 2019

Design for Assignment 7

For this assignment we were tasked to implement a program in C that reads through a text and tries to find occurrences of certain words, some of which have associated words with them. To do this we were instructed to create a Bloom Filter ADT and a Hash Table ADT that stores the key words and saves them, and looks up very rapidly if a certain word exists within the hash table. To create a hash table we first need to create the ADT for a linked list to handle collisions. This linked list was programed to store GoodSpeak values, which is similar to a dictionary that contains a key and a value, which is the translation of the word. The following functions are included in ll.h:

```
- Creates a linked list node
ListNode *ll_node_create(GoodSpeak *gs);
- Destroys a linked list node
void ll_node_delete(ListNode *n);
- Destroys a linked list
void ll_delete(ListNode *head);
- Inserts a GoodSpeak element in the linked list
ListNode *ll_insert(ListNode **head, GoodSpeak *gs);
- Explores the linked list to find a GoodSpeak element
ListNode *ll_lookup(ListNode **head, const char *key);
```

To implement this we just used common single linked list implementations for these functions. Since linked lists are such a trivial ADT, I won't explain them in this document.

Next, we need to implement the Hash Table, which is were things start to get tricky. Thankfully, Professor Darrel Long provided most of the implementation of the Hash Table, along with the

corresponding hash functions.The functions that we did have to implement for the hash table were: the destructor, insert, and lookup functions. They can be implemented in the following way:

```
ht_delete{
    for i = 0 to length -> ll_delete(list_head)
    // Deletes each list in the hash table
    free(hash table)
}
ListNode ht_lookup{
   h = hash(key)
   //
   return ll_lookup(hash_table[h],key)
}
ht_insert{
    h = hash(key)
    ll_insert(hash_table[h],GoodSpeak_element)
}
```

Once we have our hash table ADT defined we can start coding our Bloom Filter. For this implementation of a bloom filter ADT we are going to use the Bit Vector ADT that we designed for assignment 5. Again, some of the bloom filter implementation is already provided to us by Professor Darrel Long. We are asked to implement the functions that manipulate said bloom filter objects. The following functions define the delete, insert, and probe functions of the bloom filter:

```
bf_delete{
    bv_delete(bitfilter)
    free(bloomf)
}
bf_insert{
```

```
    h1 = hash(first_salt,key)
    h2 = hash(second_salt,key)
    bv_setbit(bitfilter,h1)
    bv_setbit(bitfilter,h2)
}
bf_probe{
    h1 = hash(first_salt,key)
    h2 = hash(second_salt,key)
    if h1 is in bitfilter and h2 is in bitfilter
        return true
    else return false
}
```

Since the create function is already implemented we don't have to define it ourselves.

Finally, to complete our program we need to write the main source code, which will read two files, one containing the key words with their translations and another one with just key words with no translation.

Simply, we will use getopt() functions from the getopt.h C library to parse the instructions from the command line when the excecutable is called. Then, we will use the lexical analyzer "flex" which gives us functions to read through the files and get each word.

After the files are read, we will read a third file which is the one we will be evaluating. We will store the occurrences of the key words and their translations as the program goes through the text. At the end of the program we will print out to the user a message that explains the key words found in the text, and explain if they have a translation or not.