Devan O'Boyle
CSE13s
Assignment 7: The Great Firewall of Santa Cruz

Design

Purpose: In this assignment, we will be using bloom filters, hash tables, and linked lists to catch if certain words entered by the user match words found in a badspeak or newspeak text file which contain words that the user should not say. If there are matches, then a message telling the user to use proper language will be displayed followed by a list of words that they should not say.

In the program various data structures will be used: Bloom Filters, Hash Tables, Bit Vectors, Linked Lists, and Nodes

Bloom Filters: The bloom filter will be given three salts that will be implemented in the create function. When oldspeak is inserted into the bloom filter, the three salts are used with the hash table to get three indices which will then be set in the bit vector. There is also a probe function that tells whether or not oldspeak has been added to the bloom filter or not.

Pseudocode:

```
bloom filter insert:
takes in the bloom filter and an oldspeak word
take the hash of the primary salt and the given oldspeak and mod it by the length of the bloom filter
set this bit to the bloom filter's bit vector using the previously calculated value as the index
repeat the last two steps for the secondary and tertiary salts
```

```
bloom filter probe:
takes in the bloom filter and an oldspeak word
take the hash of the primary salt and the given oldspeak and mod it by the length of the bloom filter
create a boolean variable and set it to the returned value of getting the bit of the bit vector at the index of the previously calculated value
repeat the last two steps for the secondary and tertiary salts
then check if all of the boolean variables are true
if so then return true to show that the oldspeak word was found
otherwise return false to indicate that it wasn't
```

Bit Vectors: The bit vector works similar to how it did in assignment 5. There are functions to set, clear, and get bits which are done through bit shifting.

Hash Tables: The hash table consists of an array of linked lists. There is a lookup function that hashes the oldspeak in order to find the index of the linked list to perform a lookup on. If the lookup finds the node, then the pointer to that node is returned. Otherwise, a null pointer is returned. There is also an insert function that takes in an oldspeak and its corresponding newspeak translation and then hashes the oldspeak to find the index to insert it similar to what was done in the lookup function.

Pseudocode:

```
hash table lookup:
take in the hash table and an oldspeak word
create an index variable and set it to the hash using the hash's salt and oldspeak and take the mod of it and the hash table size
if linked list at that index is null, return null
otherwise call the linked list lookup function with the index and oldspeak and return the value it returns

hash table insert:
take in the hash table, an oldspeak word, and its newspeak variant
create an index variable and set it to the hash using the hash's salt and oldspeak and take the mod of it and the hash table size
if the linked list at that index is null, create a new linked list at that index
call the linked list insert function to insert the oldspeak and newspeak
```

Nodes: Each node consists of oldspeak and its newspeak translation. An oldspeak with no newspeak translation is badspeak. This structure is only used to store nodes as a data type so it only contains create, delete, and print functions as all of these other data structures do. However, this structure's create function is a bit different since it has to duplicate the oldspeak and newspeak strings before they can be set as variables of the node. For this a string duplicate function will be implemented which is done by simply calling strcpy on the given string and allocating the memory for the length of the given string + 1.

Linked Lists: A link list consists of nodes and contains lookup and insert functions to search for and add nodes to the linked list, respectively. The lookup function iterates through the linked list and finds the index of the node containing oldspeak and returns the pointer to it. There is also a move to front option that has the node containing the oldspeak be moved to the front of the linked list. The insert function is performed by first calling on the lookup function to make sure that there isn't already a node with the same oldspeak in it. If the linked list does contain the same oldspeak, then no node is inserted. Otherwise, the new node is inserted at the front of the linked list.

Pseudocode:

```
linked list lookup:
takes in a linked list and oldspeak
increment the number of seeks
iterate through all of the nodes in the linked list
    increment the number of links
    check if the oldspeak of the current node matches the oldspeak given by the parameter
```

```
        check if move to front is on
            if so, set the current next node to the current node's previous next node
            set the current previous node to the current node's next previous node
            set the head's next node to the current's next node
            set the head's node to the current's previous node
            set the current node to the head's next previous node
            set the current node to the head's next node
        return the current node
otherwise return null to indicate the node wasn't found
```

```
linked list insert:
takes in the linked list, oldspeak, and its newspeak variant
if the oldspeak doesn't already have a node in the linked list
    then create a new node with the oldspeak and newspeak
    set the head node to the new node's previous node
    set the head node's next node to the new node's next node
    set the new node to the head's next previous node
    set the new node to the head's next node
    increment the length of the linked list
```

For the main function, there will be a list of badspeak words that will be read in first. Each badspeak word will be added to the bloom filter and hash table. Then oldspeak and newspeak word pairs will be added to the bloom filter. Then words will be read in through standard input. Check each word if it is in the bloom filter by using the bloom's probe function. If the user inserted a badspeak word, meaning that there is no newspeak translation for it, then insert the word into a list that will be printed out later. If the word is in the bloom filter but does have a newspeak translation, then that word should be sent to a list of oldspeak words with newspeak translations that will also be printed out later. Otherwise, if the word isn't in the filter, no action needs to be taken. If the user inputed both badspeak and oldspeak words with newspeak translations, then a mixspeak message is printed. If the user only inputted badspeak, then a badspeak message will be printed with the badspeak words that were entered. If the user inputted only oldspeak words with newspeak translations, then they are given a wrongthink message with proper newspeak translations. Before any prints are done though, first there should be a check to see if the user wants to print statistics instead. If so, then no messages should be printed and instead the number of seeks, the average seek length, the hash table load, and the bloom filter load should be printed.

```
Pseudocode:
getopt loop:
-h prints help message and quits
-t sets the hash table size (default 10000)
-f sets the bloom filter size (default 1048576)
-m enables move to front
-s enables printing statistics
```

```
after get opt loop
create a bloom filter and hash table using their corresponding sizes
read in input from the badspeak file and insert each word into the bloom filter and hash table, for the newspeak parameter in the hash table, null should be passed
next read in input from the newspeak file and insert each word into the bloom filter and hash table with the newspeak variant being passed into the hash table
create two separate linked list to keep track of the user’s bad words and newspeak words
create two separate boolean variables to check if the user is guilty of a thought crime or wrongthink crime
using a regular expression, read in each word and lowercase it
call the bloom filter probe function to check if the word is a badspeak or newspeak word, if it is either of those, then insert it into the corresponding linked list and set the corresponding boolean crime variable to true
after all of the words have been read in and checked, check if printing statistics is enabled
if so, print the statistics and then quit
otherwise, check if the thought crime and wrongthink crime variables are true
    if they both are true, then print a mixspeak message followed by all of the badspeak and words with newspeak variants that they said
    if only thought crime is true, then print a badspeak message followed by the badspeak words they said
    if only wrongthink crime is true, then print a goodspeak message followed by the words with newspeak variants that they said
    otherwise don’t print anything because they didn’t commit a crime
```