# ASSIGNMENT 6 DESIGN DOCUMENT

Pehara Vidangamachchi

November 11th 2022

Ethan L. Miller

## About:

This assignment aims to create a bloom filter and a hash table. By implementing such, my program should be able to filter through newspeak and bad speak and filter out any phrases using bad speak. Essentially the bloom filter acts as a space-efficient probabilistic data structure. As described in the assignment document, a bloom filter can be represented as an array of bits "m", it will utilize k different hash functions. Which then allows the filter to set an element to the filter, generating a uniform pseudo-random distribution. And in order to complete the above, they will be executed through the use of several functions such as ht.c, ll.c, node.c, bf.c, bv.c, parser.c and finally Makefile. The banhammer will then act as a filter towards citizens of our imaginary town and filter out any words deemed unfit or inappropriate.

## Design Process for ht.c

The purpose of this program is to create a hash table. Sharing similarities with the bloom filter, a hash table contains a salt that gets passed down whenever a new old speak element is added. Using chained hash tables allows us to avoid collisions. Following the pseudo-code given in the assignment, I was able to execute the code in a functional manner. Through the use of pointers and accessing information within the bloom filters structure I was able to create cohesive code that properly executed the hash tables functions.

*#include any header files*

*#declare any variables*

*Create the hash tables structure:*

*Include the salts, size, keys, hits, misses, examined, mtf and lists.*

*Create a function to create the hash table:*

*Set the hash table equal to its variable counterparts pointer*

*Which receives the size of the hash table*

*Create an if statement that sets up all the parameters of several variables*

*Set the mtf, salt, num hits, num keys, size and lists equal to their respective counterpart.*

*Create an if statement in case the hash table is not equal to the lists*

*Free the allocated memory and set it equal to null*

*Return the hash tables contents*

*Create a function to delete the hash table:*

*Free the hash table and set it equal to null*

*Create a function to create the hash tables size:*

*Return the size from the hash tables library*

*Create a function that looks through the hash tables contents:*

*Initialise any variables that will be utilized within the code*

*Create an if statement when the list is equal to null:*

*Create a list when there is no list*

*Insert the list using ll insert*

*Create a function that keeps count of the hash tables:*

*Return the hash tables lists*

*Create a function that prints the hash tables contents:*

*Initialise any variables that will be utilized within the code*

*Create a print statement to print out the hash table*

*Create a function that keeps track of the hash tables stats:*

*Initialise any variables that will be utilized within the code*

*Using pointers set the value equal to the original*

## Design Process for ll.c

The purpose of this program is to create a linked list. They are defined as having two sentinel nodes and the field mtf. The assignment document goes in depth about their functionality and purpose. So following the directions given there I am able to implement my own program regarding this segment. This was done similarly to my other programs where I used struct and allocated their associated contents within each function.

*#include any header files*

*#define any variables*

*Create a function that creates a linked list:*

*Initialise any variables that will be utilized within the code*

*Create a while loop to run given the correct parameters:*

*Create an if statement when mtf is true and a bool:*

*Filter through the nodes*

*Create a function that deletes the linked list:*

*Initialise any variables that will be utilized within the code*

*Create an if statement condition that checks for the linked list*

*Clear the memory if the statement runs true*

*Return with the empty memory*

*Create a function that gets the length of the linked list:*

*Initialise any variables that will be utilized within the code*

*Get the size of the linked list using a while loop:*

*Followed by an if statement when the conditions are true:*

*Getting the math portion of its functionality*

*Create a function that looks up the contents within the linked list:*

*Initialise any variables that will be utilized within the code*

*Create a while loop that searches for the nodes and old speak:*

*Using the number of stats and linked lists move front*

*Update the old information with the new information*

*Return the difference*

*Create a function that inserts a character within the linked list:*

*Initialise any variables that will be utilized within the code*

*Create an if statement if a node contains oldspeak:*

*Replacing it with new speak*

*And checking that the node does nto already contain a match*

*Else the node is added at the head of the list*

*Create a function that prints the contents of the linked list:*

*Initialise any variables that will be utilized within the code*

*Create a print statement to print out the linked list*

*Create a function that compiles the stats of the linked list:*

*Initialise any variables that will be utilized within the code*

*Using pointers set the value equal to the original*

*Go through each variable till they are all converted using a while loop*

*Check how the numbers of links traversed between all the other programs*

## Design Process for node.c

The purpose of this program is to create a doubly linked list that will be used to solve hash collisions, with each passing node containing a segment of old speak and its respective newspeak translation. Each node contains some pointer towards a previous node due to the

implementation of linked lists. This is done by following the assignment document's directions with its pseudo code. I once again used struct and its contents to formulate each area of the code I was tasked with writing.

*#include any header files*

*Create a function to create the node:*

*Initialise any variables that will be utilized within the code*

*Create an if statement with the Node:*

*Create a copy of old speak and new speak by allocating it memory*

*Create a next and prev function and set them equal to null*

*Return the node*

*Create a function to delete the node:*

*Free the node's memory*

*Set the node equal to null*

*Create a function to print the node:*

*Initialise any variables that will be utilized within the code*

*Create a print statement for each condition within an if statement*

## Design Process for bf.c

The purpose of this program is to create a bloom filter following the directions given to us in the assignment document. This is done by following the given pseudo code and executing

the code in such a way that creates a functioning filter. Information within the filter will be stored within an array and collect certain statistics. The filter followed a similar pattern as the rest where I used the contents of the bloom filters struct argument to help execute my code.

*#include any header files*

*#define any variables*

*Create the structure for the bloom filter:*

*Include the salts, keys, hits, missed, bits examined and the bit vector*

*Create a function that creates the bloom filter:*

*Initialise the variables used within the code*

*Create an if statement to run the code when the conditionals are true:*

*Set the bloom filter equal to the number of keys*

*Set the number of misses equal to the number of bits examined*

*Create a for a loop when an integer is within a certain range:*

*Set the salts equal to the default salts*

*Set the filter equal to the create function*

*Create an if statement in the case that filter is equal to null:*

*Free the memory for the bloom filter*

*Set it equal to null*

*Create a function for deleting the bloom filter:*

*Use the bit vector delete function to first remove that*

*Then use free to free bf of its memory*

*Set the bloom filter equal to null*

*Create a function for getting the size of the bloom filter:*

*Return the length of the bit vector for the size of the bloom filter*

*Create a function for inserting the bloom filter:*

*Create a for loop for when i is less than the number of hashes:*

*Set the hash table equal to a variable and mod it by the length of the bit vector*

*Set the bit vector equal to whatever value is returned*

*Create a function for probing the bloom filter:*

*Create a for loop for when the number of hashes is greader than i:*

*Set the hash table equal to a variable and mod it by the length of the bit vector*

*Create an if statement so when all the bits in the previous code are true*

*Return false to say that old speak was added to the filter*

*Else return true*

*Create a function for keeping count with the bloom filter:*

*Initialise any variables that will be utilized within the code*

*Get the count of bits within the bloom filter using a for loop:*

*Iterate throughthe loop and add 1 to count every time get bit is ran*

*Return the count*

*Create a function that prints out the bloom filters contents:*

*Initialise any variables that will be utilized within the code*

*Create a print statement to print out the bloom filter*

*Create an if statement to print the salts and values when greater than the number of hashes:*

*Print the result of the salts and value*

*Create a function for the bloom filters stats:*

*Initialise any variables that will be utilized within the code*

*Using pointers set the value equal to the original*

*Be sure to allovate each variable eqaul to eachother*

## Design Process for bv.c

The purpose of this program is to create a Bit Vector, the length defines the size of the bit vector. This will be used to help keep track of the number of bits. The code requires the usage of bitwise operations. following the assignment pdf, I executed my code by using the given directions and the accompanying pseudo-coed. Once again utilizing the Bit vectors structure given within the header file which I translated over to my c file.

*#include any header files*

*#define any variables*

*Create a structure for the bit vector:*

*Include the length and vector*

*Create a function to create the bit vector:*

*Assign the bit vector to a pointer*

*Create an if statement so*

*length within the bit vector is eqaul to length*

*The vector is equal to the size of a uint64*

*Else*

*Free the bit vector and set it equal to null*

*Finally return the vector at the end*

*Create a function for deleting the bit vector:*

*Free the bitvector and set it equal to null*

*Create a function for getting the length of the bit vector:*

*Return the length of the bit vector*

*Create a function for setting the bit vectors size;*

*Set the bit vector eqaul to its math counterpart which sets the ith bit*

*Create a function to clear the bit vector:*

*Set the bit vector equal to its math counterpart which clears the ith bit*

*Create a function to get the bit vectors contents;*

*Set the bit vector equal to its math counterpart which returns the ith bit*

*Create a function to print the bit vectors contents:*

*Initialise any variables that will be utilized within the code*

*Create a print statement to print out the bloom filter*

## Design Process for parser.c

The purpose of this program is to act as a regulator for the words that are spoken. It will filter through the valid words and compare them. The program will have to be compatible with almost everything and analyze them properly. I plan to implement this following the directions given within the assignment document.

*#include any header files*

*#deifne the parsers length*

*Create the structure for the parser:*

*Include the file, line and line offset*

*Create a function to create the parser:*

*Initialise any variables that will be utilized within the code*

*Construct the main code for the parser using an if statement:*

*Set p equal to f*

*Set the memory of p and set the line offset equal to zero*

*Create a function to delete a parser:*

*Free the parser and set it equal to null*

*Create a function to transition to the next word:*

*Initialise any variables that will be utilized within the code*

*Use fgets to get the current line and the max length*

*Create a while loop that goes to the next word till it reaches the end:*

*Using pointers keep transitioning towards the next word*

*When there are no more words left:*

*Return a false*

## Design Process for Makefile

The purpose of this program is to create a makefile that compiles the program into functioning code. This allows for each file to compile and work with the main function in unison. The makefile also allows the user to clean the files and wipe the created files, alongside alerting the user of any code errors.

*EXECBIN = banhammer file name*

*SOURCES = $(wildcard *.c)*

*OBJECTS = $(SOURCES:%.c=.o)*

*CC = clang*

*CFLAGS  = error message*

*LFAGS = $(shell pkg-config –libs gmp)*

*.PHONY: make it  all clean*

*$(EXECBIN): $(OBJECTS)*

*$(CC) -o $@ $^ $(LFLAGS)*

*%.o : %.c*

*Clean all files that are .o:*

*Remove said files using rm*

## Design Process for banhammer.c

The program reads a list of old speak and newspeak, alongside bad speak, which is just a translated version of old speak using newspeak. Each word checked must be added to the bloom filter and checked with the hash table. Essentially the filter is looking to conserve social cohesion and avoid public distress in its make-believe setting. There are message formats to follow in the scenario a citizen uses inappropriate language deemed as unfit due to its hurtful, offensive, unfortunate and far too descriptive language. In order to create this within my code I had to create multiple helper programs such as bloom filter which filters to see whether a word is a member of a set or not. This was accompanied by a bit vector which helped allocate memory. We were given a city file that contained hash functions to use within our code which is used for hashing shorter strings. Alongside this, we had linked lists which acted as a solution towards avoiding hash collisions with the use of nodes. Nodes are another function I created that helps differentiate between all the different words and simplifies the implementation of the link list.

Finally leaving me with the parser which acted as a way to parse out the words that the citizens speak and transform them into an input stream. All of these functions combine to create the main program which then executes the task given.

*#include any header files*

*Create the main function*

*Initialize any variables that will be utilized within the program*

*Create a while loop for the options within the code:*

*Parse through each case and accompany each with the respective function they execute*

*Innit of the bloom filter followed by the hash table to use later*

*Read the files using fopen for bad speak and newspeak*

*Create a character for badspeak, newspeak and oldspeak*

*While loop that runs till it reaches the end of the file:*

*Using bloom filter insert and hash table insert for bad speak*

*Close the bad speak file at the end*

*Create a while loop that runs till the end of the file again:*

*This time insert the old speak and newspeak*

*Close the new speak file at the end*

*Read the stdin file*

*Set each variable equal to its respective counterpart, be sure to initialize everything*

*Set the line size equal to getline function*

*Parse through the words using fopen*

*Use the parser create function created*

*Create a while loop that runs while true*

*If statement if there is no next word:*

*Break from the code*

*Else if statement that parses the words through bloom filter:*

*Create a temp variable and set that equal to the hash table look up*

*Create an if statement if hash table lookup sets the newspeak eqaul to null:*

*Set the thoughtcrime equal to one*

*Then set the memory for the thought crime list and parse through the words*

*Else statement*

*When no thoughtcrime has been completed set it to true*

*Allocate memory towards the oldspeak list*

*If statement that prints the actions for thoughtcrime and rightspeak:*

*Print the message provided*

*Set the word number equal to zero*

*Create a while loop for thought crim:*

*When thought crime does not equal zero print the thought Crime statement and add to the word number*

*Create a while loop for oldspeak:*

*When oldspeak does not equal zero return its print Statement*

*Create an if statement for when thoughtcrime is committed and*

*There is no rightspeak*

*Print the badspeak message*

*Create a while loop when the thoughtcrime count is not zero:*

*Print the thoughtcrime*

*Add to the word number*

*Create an if statement for when though crime is not committed and*

*There is rightspeak*

*Print the goodspeak message*

*Create a while loop when the rightspeak count is not zero:*

*Print the rightspeak*

*Add to the word number*

*Create floats for each of the statistics*

*Create an if statement to print each stat:*

*Using if statements for each individual stat print their contents*

*Free memory at the end of the program*

*Return zero*