Node.c

Create(oldspeak, newspeak)

Malloc node,
Set oldspeak and newspeak to inputted versions
Set prev and next to NULL

Delete

Free node
Set given pointer to null

Print

If newspeak exists, print(oldspeak -> newspeak) else print (oldspeak)

LinkedList.c

Struct LinkedList

Var length;
Node head;
Node Tail;
Bool MTF;

Create(Bool mtf)

LL = Malloc space for Linked List,
Set LL mtf to given mtf

Create head and tail sentinel node

```
Set LL head and tail to the previously created sentinel nodes
Return LL
```

Delete(LL)

```
Current node = head of LL

While (current node != NULL)
    Next node = current node-> prev
    Free(current node)
    Current node = next node

free(LL)
Set LL to NULL
```

Lookup(LL, oldspeak)

```
Iterate over all nodes in LL
    If current node's oldspeak == oldspeak
        If mtf is true,
            Move node to front
        Return the node

Else return NULL
```

Insert(old, new)

```
Create new node called in using (old,new)



head->prev = in
```

Node after head’s->next = in

in->next = head
in->prev = node after head

Increment linked list length

Print()

```
Loop through linked list
    For each node call
        node_print();
```

Stats(seeks, links)

Set seeks = ll->seeks
Set links = ll->links

BitVector.c

Struct bitvector

```
Uint32 length
Uint64 *vector
```

Create(length)

```
Int blocks = length/64 + 1;
Malloc space for bitvector called bv
calloc (blocks, sizeof(uint64_t)) called vector

bv->length = length
bv->vector = vector
```

Delete(**bv)

```
Free bv->vector
Free bv

Set bv to NULL
```

Set_bit(i)

```
Block = i /64
Bit = i % 64

Set bit in block Block and position i
bv->vector[block] = bv->vector[block] | (1<<bit)
```

Get_bit

```
Block = i/64
Bit = i%64

//Shift bit all the way to the right then all the way to the left

Result = bv->vector[block] << (64-1)-bit;
Result = result >> 63

Return (uint8_t) result;
```

Print()

```
For (int i = 0; i < bv->length; i++);
{
    Current_bit = get_bit(i)
    if (current_bit == 0)
        Print 0
    Else
        Print 1

}
```

BloomFilter.c

Struct

From assignment PDF
Variables keys, salts, hits, misses, bits examined
Bitvector* filter

Create

Use create from assignment PDF

Delete

Call bv_delete(bf->filter)
free(bf)
Bf = Null

Size

Return bf->filter->length

Insert(oldspeak)

```
    For every salt, hash oldspeak with salt
        Get %size of hash and set that bit
```

Probe(oldspeak)

```
    For every salt, hash oldspeak with salt
        Get %size of hash and check if that bit is ==1
        If its not, return false

    Return true
```

Count()

```
Counter = 0
For ( i = 0; i < bf_size; i++)
    If bit at i location is 1, increment counter

Return counter
```

Print

```
    Call print on underlying bitvector
```

Stats

```
    Set all inputs to their respective stored values
```

HashTable.c

Struct

Get struct from assignment pdf, should have vars
Salt, size, keys, hits, misses, times examined, bool mtf
Double pointer to linked lists

Create

Use create from assignment pdf

Delete

Loop through every linked list and call ll_delete() on it

Free linked list that had linked lists as nodes
Free(hash table)
Set hash table to NULL

Size

Return ht->size

Lookup(oldspeak)

Hash oldspeak with salt and %size to get index

Call ll_lookup with that index linked list

Return Node if it exists
Return NULL if it does not

Insert(old, new)

Hash oldspeak and % 64 to get index
Call ll_insert on list[index]

Print

Iterate over every list and call ll_print on it

Parser.c

Struct

Use struct from pdf

Create

Allocate memory for parser struct
Set p->f to inputted file

```
Set line offset to MAX_PARSER LENGTH  +1
```

Delete(p)

```
Close the file in p
Free(p)
Set *p to null
```

Next_word(p, word)

```
If (line offset > max line length)
    Read the next line, if no new line exists return FALSE

While(there is not a valid character)
    Increase the offset
    If you reach the end of the line, read the next line
Int i = 0;
While (there is a valid letter starting from the pointer)
    Set word[i] = character
    Increment i and offset

Set word[i]=0 to signal end of string
Return TRUE
```