Bv.c

(Must be in bv.c)
struct BitVector {
	uint32_t length ;
	uint8_t * vector ;
};

*BitVector *bv_create(uint32_t length)* {
	Use calloc or malloc to allocate memory for the Bitvector
	Use calloc or malloc again to allocate memory for the array size length
	(probably calloc to make each bit in array 0)
	Return pointer
}

*Void bv_delete(BitVector **bv)* {
	Free array that is size length
	Free btivector
	Set pointer to NULL
}

Uint32_t bv_length(bv) {
	Return the bit vector length
}

Bool bv_set_bit(bv, uint32_t i) {
	Set the i_th bit of the bit vector. If i is out of range of the bitvector return false;
}

*bool bv_clr_bit(BitVector *bv, uint32_t i)* {
	Same as bv_set_bit but now clears the i_th bit of the bitvector
}

*bool bv_get_bit(BitVector *bv, uint32_t i)* {

Same as clear and set bit, returns false if bit is 0 and true if bit is 1

}

Node.c

*Node *node_create(char *oldspeak, char *newspeak)* {

Allocate memory using calloc or malloc

We need to make a copy of oldspeak and newspeak using strdup() (#include string.h)

Set left and right to NULL

Return the pointer to the Node

}

*void node_delete(Node **n)* {

Free only Node n, not the next and previous nodes.

Because we allocated memory for oldspeak and newspeak, we much free those too

}

*void node_print(Node *n)* {

If node n has oldspeak AND newspeak use:

printf ("%s -> %s\n", n- > oldspeak , n- > newspeak) ;

If node n ONLY has oldspeak, use

printf ("%s\n", n- > oldspeak) ;

}

Bf.c (BloomFilter)

```
struct BloomFilter {
  uint64_t primary[2];   // Primary hash function salt.
  uint64_t secondary[2]; // Secondary hash function salt.
  uint64_t tertiary[2];  // Tertiary hash function salt.
  BitVector *filter;
};
```

(must go in bf.c)

*BloomFilter *bf_create(uint32_t size)* {
    Allocate memory for the Bloom Filter
    Set primary[0] to the lower primary salt from salts.h
    Set primary[1] to the higher primary salt from salts.h
    Do the same for secondary and tertiary salts
    Use bv create to make filter
}

*void bf_delete(BloomFilter **bf)* {
    Free filter using bv_delete
    Free bf
    Pointer Bf = NULL
}

*uint32_t bf_size(BloomFilter *bf)* {
    Use bv_length to get size of the bloom filter
}

*void bf_insert(BloomFilter *bf, char *oldspeak)* {
    Use bv_set_bit to insert, use hash function from speck.c together with each Salts, make sure it is in bounds of the bloom filter.
}

*bool bf_probe(BloomFilter *bf, char *oldspeak)* {
    The same inserting but this time using bv_get_bit
}

*uint32_t bf_count(BloomFilter *bf)* {
    Have a uint32_t variable to hold the count
    Iterate through from 0 to length of bloom filter
        Increment the variable
    Return the variable
}