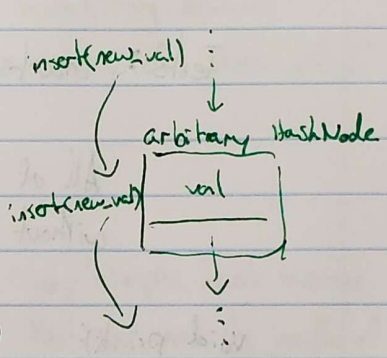
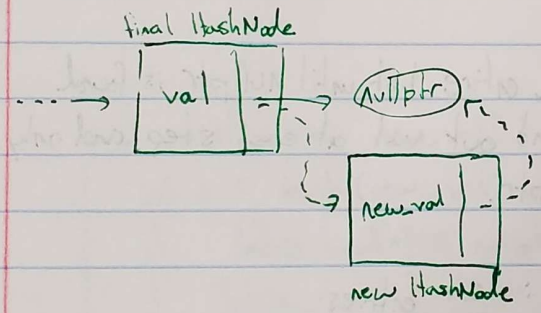
HashNode: Member variables: string val
HashNode* next

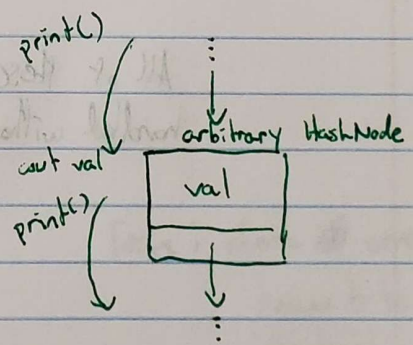
Member functions: HashNode(string new_val)
recursive { void insert(string new_val)
void print()

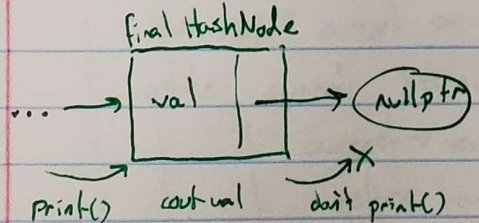
```
void insert(string new_val):
    if next != nullptr
        next->insert(new_val)
    else
        next = new HashNode(new_val)
```






```
void print():
    cout val
    if next != nullptr
        next->print()
```





void insert(string new_val):

The idea is to iterate through the list until the end (once it finds nullptr). Then, change next to a new HashNode with the desired value.

Tests: insert a string with the value... "nullptr"
"NULL"
""
"hello"

All of these should be handled without issue.

void print():

Again, iterate through the entire list until nullptr is found. However, this time we print out val at each step and only stop once next == nullptr.

Tests: print a list with: 2+ entries
1 entry
0 entries.

All of these should be handled without issue.

HashTable: Member variables: int size
HashNode** table_head

Member functions: HashTable()
void insert(string new_val)
bool contains(string new_val)
int hash(string new_val)

variable table_head is a pointer to an array of HashNode*, size 'size'

int hash(string new_val):
    return new_val.size() % size

very simple hash function that returns the length of the string modded by size.

void insert(string new_val):
    int n = hash(new_val)
    if table_head[n] == nullptr
        table_head[n] = new HashNode(new_val)
    else
        table_head[n] -> insert(new_val)

Tests: insert "", "nullptr", "hello"

bool contains(string new_val):
    int n = hash(new_val)
    HashNode* current = table_head[n]
    while current != nullptr
        if current->val == new_val
            return true
        current = current->next
    return false

Tests: check something known to be in list, top level

check something known to be in list, mid level

check something known not to be in list.