Complexity:

The complexity of a hashtable depends on the number of collisions. Ideally, we want our hash function to have little to no collisions. Worse case, the complexity will be **O(n).** If there is no collision, then the hashtable complexity will be **O(1).**

Hash Function:

```
unsigned long HashTable::hash_function(const std::string &key) const {
    unsigned long hash = 0;
    for (char c : key) {
        if (c >= 'a' && c <= 'z') {
            hash += c - 'a' + 1;
        }
    }
    return hash % CAPACITY;
}
```

Insert Function:

```
void HashTable::insert(const std::string &key, int value) {
    unsigned long index = hash_function(key);
    for (auto &pair : table[index]) {
        if (pair.first == key) {
            pair.second = value;
            return;
        }
    }
    table[index].emplace_back(key, value);
}
```

Contains Function:

```cpp
bool HashTable::contains(const std::string &key) {
    unsigned long index = hash_function(key);
    for (auto &pair : table[index]) {
        if (pair.first == key) {
            return true;
        }
    }
    return false;
}
```

Testing:

Hash Function:

```cpp
void test_hash_function() {
    HashTable ht;

    char str1[] = "juan";      // 10 + 21 + 1 + 14 = 46
    char str2[] = "gisselle";  // 7 + 9 + 19 + 19 + 5 + 12 + 12 + 5 = 88
    char str3[] = "alex";      // 1 + 12 + 5 + 24 = 42

    std::cout << "Hash of '" << str1 << "': " << ht.hash_function(str1) << std::endl;
    std::cout << "Hash of '" << str2 << "': " << ht.hash_function(str2) << std::endl;
    std::cout << "Hash of '" << str3 << "': " << ht.hash_function(str3) << std::endl;
}
```

Insert Function:

```
void test_insert_function() {
    HashTable ht;

    //Create a set of key-values pairs to test
    std::vector<std::pair<std::string, int>> key_values_pair = {
        {"juan", 46},
        {"gisselle", 88},
        {"alex", 42},

    };

    // Insert key-value pairs into the hashtable
    for (const auto &pair : key_values_pair) {
        ht.insert(pair.first, pair.second);
    }

    // Check if inserted keys are present in the hashtable
    for (const auto &pair : key_values_pair) {
        if (ht.contains(pair.first)) {
            std::cout << "Key '" << pair.first << "' with value " << pair.second << " was inserted correctly." << std::endl;
        } else {
            std::cerr << "Error: Key '" << pair.first << "' was not found in the hashtable." << std::endl;
        }
    }
}
```

Contains Function:

```
void test_contains_function() {
    HashTable ht;

    ht.insert("juan", 46);
    ht.insert("gisselle", 88);
    ht.insert("alex", 42);

     // Test cases
    std::vector<std::string> test_keys = {"juan", "gisselle", "alex"};


    for (const auto &key : test_keys) {
        if (ht.contains(key)) {
            std::cout << "Key '" << key << "' is present in the hashtable." << std::endl;
        } else {
            std::cerr << "Error: Key '" << key << "' is not present in the hashtable." << std::endl;
        }
    }
}
```

Overwrite Function:

```
void test_overwrite_collision() {
    HashTable ht;


    ht.insert("juan", 1);


    ht.insert("juan", 2);


    if (ht.contains("juan")) {
        int value = ht.get("juan");
        if (value == 2) {
            std::cout << "Key 'juan' was correctly overwritten with value 2." << std::endl;
        } else {
            std::cerr << "Error: Key 'juan' has incorrect value " << value << std::endl;
        }
    } else {
        std::cerr << "Error: Key 'juan' was not found in the hashtable." << std::endl;
    }
}
```