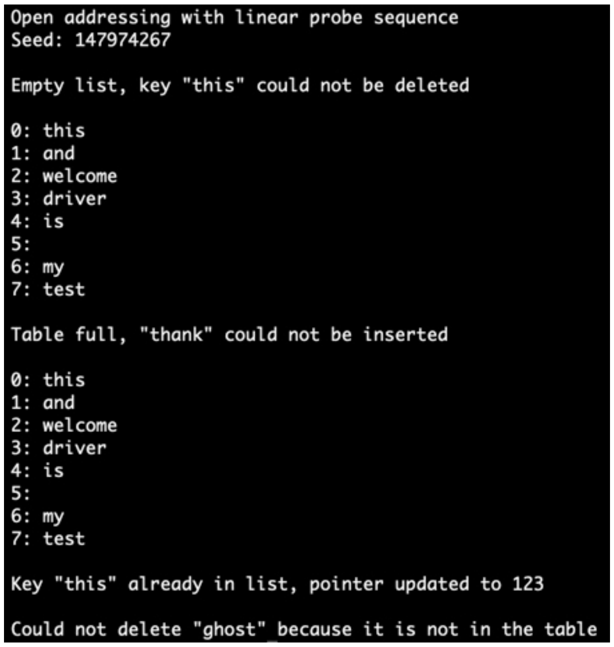
Test Plan – MP7
Cameron Burroughs
ECE 2230
Dr. Russel

UNIT DRIVER 1

Introduction:

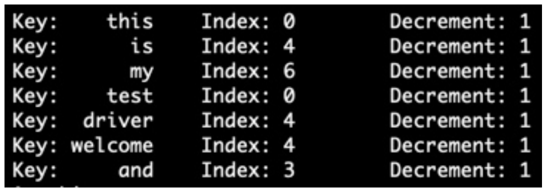
The first unit driver, void UD1(void), tests boundary and other conditions on a small, even sized table. It begins by constructing a table and attempting to delete from the table before any entry is added. This attempt should return a NULL from the delete function but should not cause any assertion or boundary failures. Next, 7 items are inserted into the list according to the index determined by the hash function, and the table is printed. A printf statement contained in the table_insert function can be uncommented to better demonstrate the collision resolution. Next, an additional key is inserted into the list. For the linear probe type, this should result in a message that the key could not be inserted. Since the separate chaining type cannot fill up, this item should also be inserted to the list. The table is then printed again, for linear and double, this result should be unchanged from the first print. For separate chaining, an additional key should be present.

Linear probing results:



```
Open addressing with linear probe sequence
Seed: 147974267

Empty list, key "this" could not be deleted

0: this
1: and
2: welcome
3: driver
4: is
5:
6: my
7: test

Table full, "thank" could not be inserted

0: this
1: and
2: welcome
3: driver
4: is
5:
6: my
7: test

Key "this" already in list, pointer updated to 123

Could not delete "ghost" because it is not in the table
```

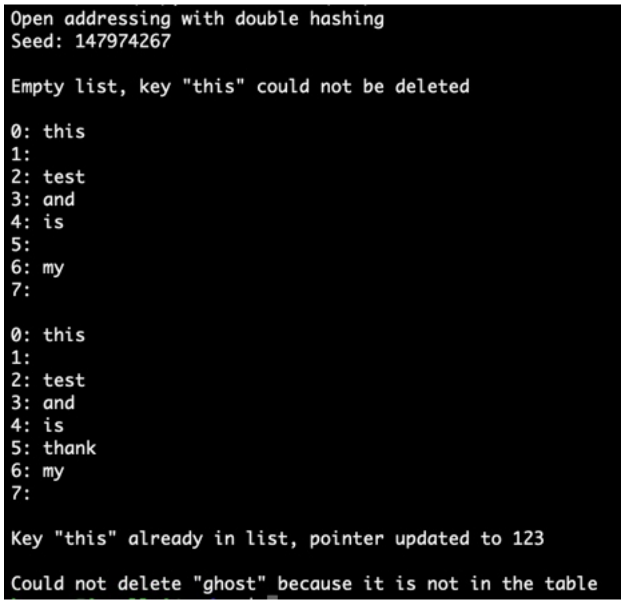
Above are the results from unit driver 1 (./lab7 -u 1) for linear probing. As you can see, when the program attempted to delete key "this", a NULL pointer was returned, and the program continued running. Next, we can see that the table is completely filled with 7 items in the subsequent print. The next photo shows how collision resolution occurs.



```
Key:    this    Index: 0        Decrement: 1
Key:      is    Index: 4        Decrement: 1
Key:      my    Index: 6        Decrement: 1
Key:    test    Index: 0        Decrement: 1
Key:  driver    Index: 4        Decrement: 1
Key: welcome    Index: 4        Decrement: 1
Key:     and    Index: 3        Decrement: 1
```

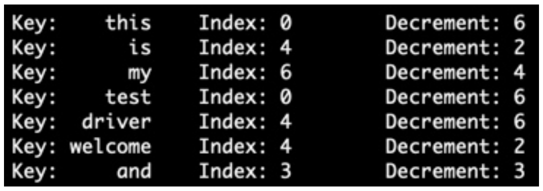
First, “this” is inserted and, using the hash function, its index is determined to be 0. Because the list is currently empty, it is inserted to index 0. Next, “is” is inserted at index 4 and “my” at index 6. When key “test” is inserted, it first attempts to be inserted at index 0, however this position is already filled, so it decrements its index by 1 until an empty position is found. Because 0 is at the start of the list, it wraps to the bottom of the list (position 7). Since this position is not empty, test is inserted at position 7. Next, both “driver” and “welcome” collide at position 4, which is already filled. The “driver” key’s index is decremented to 3 and inserted into the empty position. When “welcome” is attempts to be inserted at index 4, it decrements to three, then again to position 2, where it is finally inserted. Finally, “and” tries to insert at position 3, but decrements until it finds its place at position 1. When another key “thank” is attempted to be inserted, the table is now full, and it is not inserted. The following table print is identical to the original. Finally, key “ghost”, which is not present in the list, is attempted to be deleted. As it is not present, nothing happens, and the code exits.

Double Hashing Results:



```
Open addressing with double hashing
Seed: 147974267

Empty list, key "this" could not be deleted

0: this
1:
2: test
3: and
4: is
5:
6: my
7:

0: this
1:
2: test
3: and
4: is
5: thank
6: my
7:

Key "this" already in list, pointer updated to 123

Could not delete "ghost" because it is not in the table
```

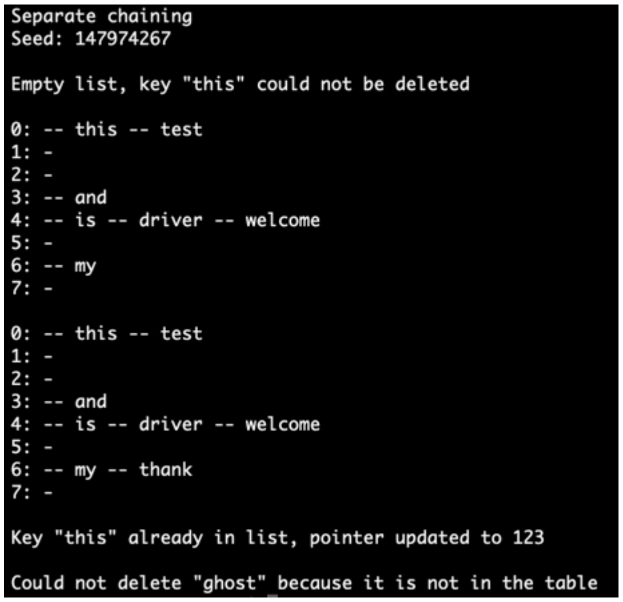
The results for double hashing differ from linear probing, because its decrement value is different. For a table size of 8, the table never fills up using these specific keys. This is because the secondary hashing function will often times create a decrement value that will not reach every position in an even-sized table.



```
Key:    this    Index: 0        Decrement: 6
Key:      is    Index: 4        Decrement: 2
Key:      my    Index: 6        Decrement: 4
Key:    test    Index: 0        Decrement: 6
Key:  driver    Index: 4        Decrement: 6
Key: welcome    Index: 4        Decrement: 2
Key:     and    Index: 3        Decrement: 3
```

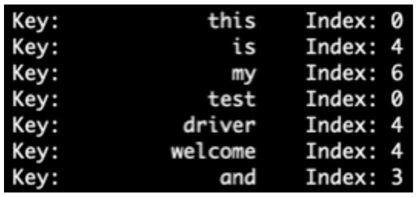
First, “this” is inserted at position 0, “is” at position 4, and “my” at position 6. When “test” is inserted, it causes a collision and it decrements by a value of 6, leaving it to be inserted at empty position 2. Next, “driver” collides at position 4. It again decrements by a value of 6, causing it to collide again at position 6. It decrements again to position 0, where another collision occurs and another decrement. It then collides again at position 2, decrements, and returns back to position 4, where it then exits. Because of the even table size and even decrement value, the key never reached the empty positions of 1, 5, or 7 and could not be inserted. Next, “welcome” is attempted to be inserted with a decrement value of 2. Again, this key collides at every other position until it reaches its starting point. Because it never hit the empty odd positions, it was never inserted. When “and” is inserted, the list is not full and position 3 is empty, so it inserts without issue. Because the list is not full, an additional key “thank”, that could not be inserted in the linear probing test, can now be placed in position 5.

Separate Chaining Results:



```
Separate chaining
Seed: 147974267

Empty list, key "this" could not be deleted

0: -- this -- test
1: -
2: -
3: -- and
4: -- is -- driver -- welcome
5: -
6: -- my
7: -

0: -- this -- test
1: -
2: -
3: -- and
4: -- is -- driver -- welcome
5: -
6: -- my -- thank
7: -

Key "this" already in list, pointer updated to 123

Could not delete "ghost" because it is not in the table
```

For the separate chaining method, the list can never fill up and collisions are quickly resolved. Each time a collision occurs, the newly added key is simply added to the end of the linked list.



```
Key:          this    Index: 0
Key:            is    Index: 4
Key:            my    Index: 6
Key:          test    Index: 0
Key:        driver    Index: 4
Key:       welcome    Index: 4
Key:           and    Index: 3
```

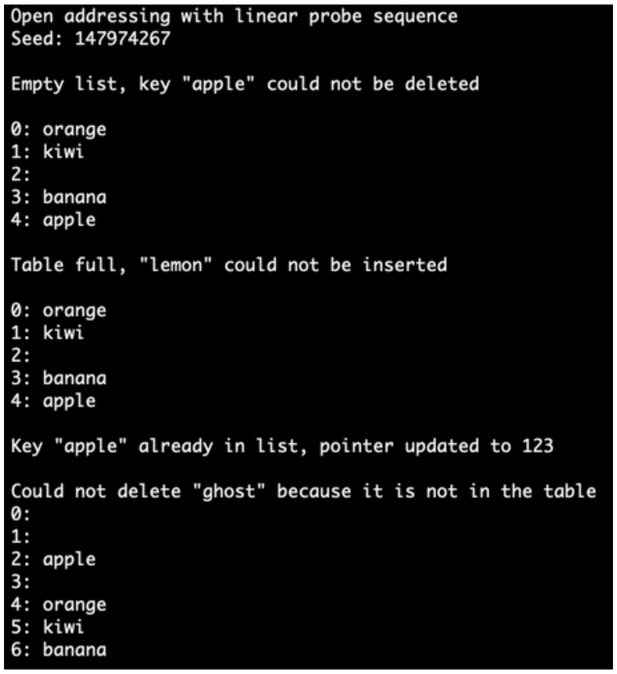
First, “this”, “is”, and “my” are inserted at their respective indexes with no collisions. Once a collision occurs when “test” is added at index 0, it is simply moved to the end of the linked list. This repeats for the collisions of “driver” and “welcome” at position 4. The resulting print can be seen above. At this point, the table would be “full” by the open addressing standards, however, a linked list representation cannot fill up and the 8th entry “thank” is added without issue.

UNIT DRIVER 2

Introduction:

This driver is similar to unit driver 1, however it tests a list with a small prime number (5) and also a rehashed prime table size of 7.

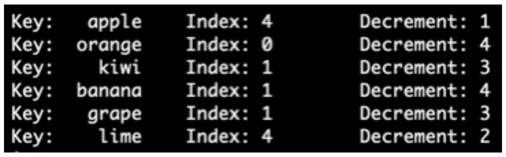
Linear probing results:



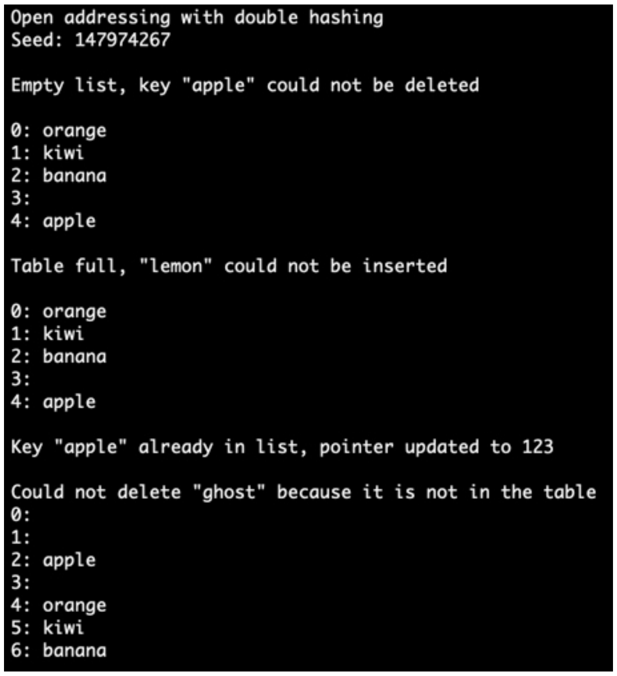
```
Open addressing with linear probe sequence
Seed: 147974267

Empty list, key "apple" could not be deleted

0: orange
1: kiwi
2:
3: banana
4: apple

Table full, "lemon" could not be inserted

0: orange
1: kiwi
2:
3: banana
4: apple

Key "apple" already in list, pointer updated to 123

Could not delete "ghost" because it is not in the table
0:
1:
2: apple
3:
4: orange
5: kiwi
6: banana
```

For the prime-sized list, all entries are inserted without issue until full, and again a 5th entry, "lemon", cannot be inserted into the list. When a duplicate is inserted, its pointer is updated, and when a non-present key is deleted, a message is displayed. When the table is re-hashed to size 7, all entries are present.

Double Hashing Results:



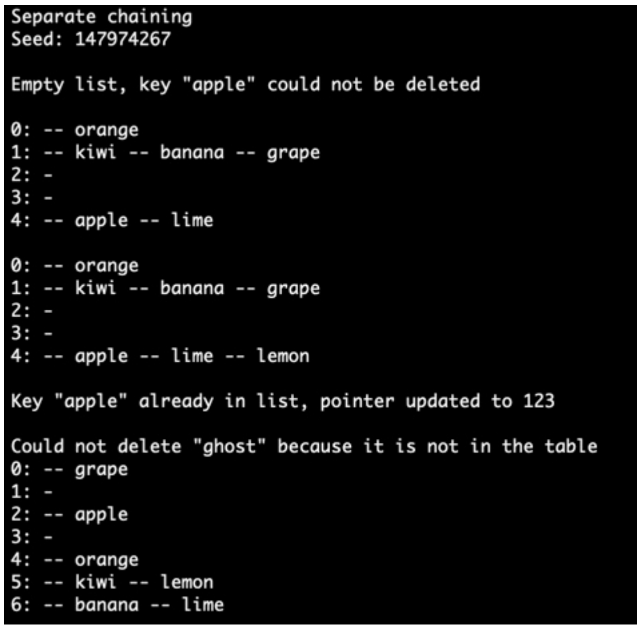
```
Key:   apple    Index: 4        Decrement: 1
Key:  orange    Index: 0        Decrement: 4
Key:    kiwi    Index: 1        Decrement: 3
Key:  banana    Index: 1        Decrement: 4
Key:   grape    Index: 1        Decrement: 3
Key:    lime    Index: 4        Decrement: 2
```



```
Open addressing with double hashing
Seed: 147974267

Empty list, key "apple" could not be deleted

0: orange
1: kiwi
2: banana
3:
4: apple

Table full, "lemon" could not be inserted

0: orange
1: kiwi
2: banana
3:
4: apple

Key "apple" already in list, pointer updated to 123

Could not delete "ghost" because it is not in the table
0:
1:
2: apple
3:
4: orange
5: kiwi
6: banana
```

For double hashing, as you can see, the table completely fills up despite collisions and varying decrement values (even and odd). Again, once full, “lemon” cannot be inserted, and “apple” can also not be inserted as it is a duplicate. When the table is rehashed, again all entries are present.

Separate Chaining Results:



```
Separate chaining
Seed: 147974267

Empty list, key "apple" could not be deleted

0: -- orange
1: -- kiwi -- banana -- grape
2: -
3: -
4: -- apple -- lime

0: -- orange
1: -- kiwi -- banana -- grape
2: -
3: -
4: -- apple -- lime -- lemon

Key "apple" already in list, pointer updated to 123

Could not delete "ghost" because it is not in the table
0: -- grape
1: -
2: -- apple
3: -
4: -- orange
5: -- kiwi -- lemon
6: -- banana -- lime
```

For separate chaining, all items are present in the table and the number of table entries out numbers the size of the table because it can never fill up. Collisions simply extend the horizontal linked-list and all items are present and properly handled after the table is rehashed.