Lab 11 – Hash Tables

Ember Roberts, Maggie Lyon, Nyla Spencer

In this lab the concepts include template class hash tables. The creation, usage, and knowledge of hash tables are important to developing a career in computer science because they are especially efficient at data retrieval. Hash tables use map keys which results in fast search completion. Hash tables also can be more memory-efficient than other data structures. These characteristics make hash tables a widely used structure in computer science.

In Task 1, a template class hash table that uses linear probing. The methods implemented include a constructor, destructor, hash (private function that accepts a string and returns an int. For this, we are going to compute the sum of all the ASCII values for the passed in string and then perform the modulus operation by the maximum size of the table), additem, removeitem, getitem, and getlength.

In Task 2, we reused the class from Lab 8, Task 2 and used it as the Item to be stored in the HashTable.

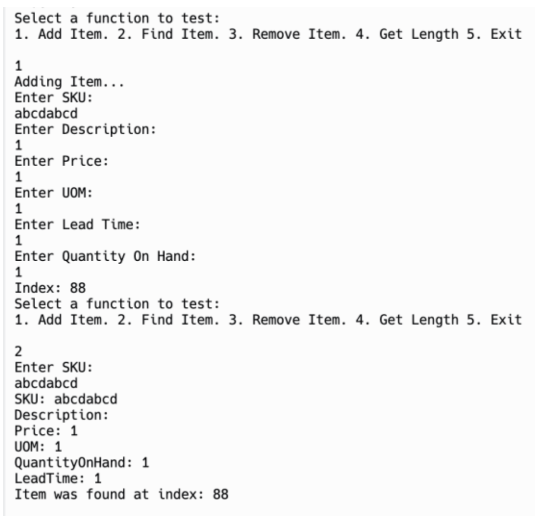
In Task 3, we created a test program like Lab 8, Task 3, to test the HashTable class for each functionality in Task 1.

In Task 4, we created a derived class from our HashTable class of Task 1 that implements a chained hash table. We modified our linked list class from Lab 8, Task 1 to be used for the chains.

Finally, in Task 5, we measured the performance of the linear probing and chained linked implementations of a hash table. We modified both classes to keep track of the number of times an item is compared in the chain for the chained linking and the number of times an additional index is checked for linear probing for the getitem. A test that adds 50 randomly generated SKU's to both hash tables from Task 1 and Task 4, then calls GetItem on the same 50 generated SKUs was created. We have recorded and displayed the total number of comparisons the Hash Table needed to complete the 50 operations here:

This operation was repeated on hash tables of size of 150 and 250. The table below shows how many comparisons were needed for each type of hash table and for each of the sizes. As the size of the tables increase, they become more efficient, because there is a higher chance that the object will be placed in an empty spot rather than a collision occuring. In addition, it is clear that the chained table is much more efficient. This is because if a collision occurs, it only needs to move down the chain one or two times ot find an empty spot, wheras the linear probing must continue to compare to the next item until it finds an empty spot, which may take many iterations.

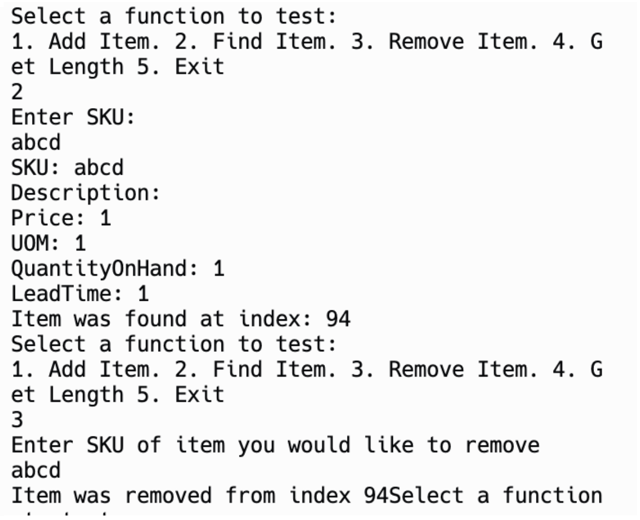
Table Size:	100 Items	150 Items	250 Items
Linear Probing	90	72	57
Chained Table	31	21	13



```
Select a function to test:
1. Add Item. 2. Find Item. 3. Remove Item. 4. Get Length 5. Exit

1
Adding Item...
Enter SKU:
abcdabcd
Enter Description:
1
Enter Price:
1
Enter UOM:
1
Enter Lead Time:
1
Enter Quantity On Hand:
1
Index: 88
Select a function to test:
1. Add Item. 2. Find Item. 3. Remove Item. 4. Get Length 5. Exit

2
Enter SKU:
abcdabcd
SKU: abcdabcd
Description:
Price: 1
UOM: 1
QuantityOnHand: 1
LeadTime: 1
Item was found at index: 88
```



```
Select a function to test:
1. Add Item. 2. Find Item. 3. Remove Item. 4. G
et Length 5. Exit
2
Enter SKU:
abcd
SKU: abcd
Description:
Price: 1
UOM: 1
QuantityOnHand: 1
LeadTime: 1
Item was found at index: 94
Select a function to test:
1. Add Item. 2. Find Item. 3. Remove Item. 4. G
et Length 5. Exit
3
Enter SKU of item you would like to remove
abcd
Item was removed from index 94Select a function
```

The screenshots above demonstrate the functionality of the main program. On the left, an item is added and then located, and on the right, an item is located and then removed.

Contributions:

Ember – Task 4, Task 5

Maggie – Task 1, Task 2, Task 3

Nyla – Lab report, Task 5