Lab 10 Report:

By: Sam Graler and Randy Hucker (Group 25)

Instructions for Running / Compiling:
.cpp files for all Tasks are located in the submitted visual studio project. Open / run the visual studio project as normal. .cpp testing files for tasks 3-5 are all included in the same project. In order to run one of them, the other two must be commented out so only one main function exists in the project. There is a note about this at the top of each .cpp file it concerns. You only must remove / add the top block comment characters "/*".

Work was divided evenly between each group member (50% for Sam and 50% for Randy)

Write a Lab Report that includes the following information:

1. **A description of the objectives/concepts explored in this assignment including why you think they are important to this course and a career in CS and/or Engineering.**

During this assignment, we implemented hash tables and chained hash tables (using linked lists). Hash tables are useful in computer science because they provide an efficient way to store and retrieve data. A hash table is a data structure that uses a hash function to map keys to values. This means that given a key, the hash function computes an index into an array of values where the corresponding value can be found. In this assignment, we used a hash method that we developed in class to convert a string value (a SKU number in our case) to ASCII, which we then summed together and modded to give us a unique hash index which reduces the case of identical hash value placements. We used hash tables for sorting/retrieval because they are good for operations that involve searching, inserting, and deleting data in constant time, on average. This is because the hash function allows for fast lookup of values based on their keys. Hash tables have an average case time complexity of O(1) for these operations.

2. **The sections from each task indicated to be included in the lab report.**

<u>Task 3: Include a screenshot of some of this testing in your lab report.</u>

These tests were performed back to back to test all functions.

```
> 1

Please Add Info to Create a Part:
SKU > 123

Price > 35

Quanity on Hand (optional, enter 0 to skip) > 3

Lead Time (days to order) > 2

Description > Item1

Unit of measure (ft, lb, per, etc...) > ft


0: Exit Testing
1: AddItem(Part*)
2: RemoveItem(Part*)
3: GetItem(Part*)
4: GetLength()
5: IsEmpty()
> 1

Please Add Info to Create a Part:
SKU > 456

Price > 43

Quanity on Hand (optional, enter 0 to skip) > 2

Lead Time (days to order) > 1

Description > Item2

Unit of measure (ft, lb, per, etc...) > ft
```

```
4: GetLength()
5: IsEmpty()
> 2

Enter the SKU number of the part you'd like to retrieve / locate. This must be a part you have already created.
> 123

SKU: 123 Desc: Item1

0: Exit Testing
1: AddItem(Part*)
2: RemoveItem(Part*)
3: GetItem(Part*)
4: GetLength()
5: IsEmpty()
> 3

Enter the SKU number of the part you'd like to retrieve / locate. This must be a part you have already created.
> 456

SKU: 456 Desc: Item2

0: Exit Testing
1: AddItem(Part*)
2: RemoveItem(Part*)
3: GetItem(Part*)
4: GetLength()
5: IsEmpty()
> 2

Enter the SKU number of the part you'd like to retrieve / locate. This must be a part you have already created.
> 456

SKU: 456 Desc: Item2
```

```
0: Exit Testing
1: AddItem(Part*)
2: RemoveItem(Part*)
3: GetItem(Part*)
4: GetLength()
5: IsEmpty()
> 4

Your List has 0 parts in it.

0: Exit Testing
1: AddItem(Part*)
2: RemoveItem(Part*)
3: GetItem(Part*)
4: GetLength()
5: IsEmpty()
> 5

Your list is empty.

0: Exit Testing
1: AddItem(Part*)
2: RemoveItem(Part*)
3: GetItem(Part*)
4: GetLength()
5: IsEmpty()
>
```

Task 4: Include a screenshot of some of this testing in your lab report.

```
3: GetItem(Part*)
4: GetLength()
5: IsEmpty()
> 1

Please Add Info to Create a Part:
SKU > 234

Price > 43

Quanity on Hand (optional, enter 0 to skip) > 4

Lead Time (days to order) > 2

Description > Item1

Unit of measure (ft, lb, per, etc...) > ft


0: Exit Testing
1: AddItem(Part*)
2: RemoveItem(Part*)
3: GetItem(Part*)
4: GetLength()
5: IsEmpty()
> 1

Please Add Info to Create a Part:
SKU > 567

Price > 43

Quanity on Hand (optional, enter 0 to skip) > 2

Lead Time (days to order) > 3

Description > Item2

Unit of measure (ft, lb, per, etc...) > ft
```

```
> 2

Enter the SKU number of the part you'd like to retrieve / locate. This must be a part you have already created.
> 234

SKU: 234 Desc: Item1

0: Exit Testing
1: AddItem(Part*)
2: RemoveItem(Part*)
3: GetItem(Part*)
4: GetLength()
5: IsEmpty()
> 3

Enter the SKU number of the part you'd like to retrieve / locate. This must be a part you have already created.
> 567

SKU: 567 Desc: Item2

0: Exit Testing
1: AddItem(Part*)
2: RemoveItem(Part*)
3: GetItem(Part*)
4: GetLength()
5: IsEmpty()
> 2

Enter the SKU number of the part you'd like to retrieve / locate. This must be a part you have already created.
> 567

SKU: 567 Desc: Item2
```

```
> 4

Your List has 0 parts in it.

0: Exit Testing
1: AddItem(Part*)
2: RemoveItem(Part*)
3: GetItem(Part*)
4: GetLength()
5: IsEmpty()
> 5

Your list is empty.

0: Exit Testing
1: AddItem(Part*)
2: RemoveItem(Part*)
3: GetItem(Part*)
4: GetLength()
5: IsEmpty()
>
```

Task 5: Include a table of the results from the 4 trials for the 2 different classes in your lab report.

Class / Number of Parts	**100**	**150**	**200**	**250**
Standard Hash Table	3726	9268	17242	27680
Chained Hash Table	332	670	1125	1730

3. **Include a discussion of what modifications you needed to make for Task 2 and Task 3.**

For Task 2 we were required to overload the string cast. The overload function we used returned a string of the SKU so that we could use it in the hash function to decide on an index. Task 3 was a bit more intricate. We needed a way to ensure that each Part object was held properly and that we could add or remove those parts as needed. We kept most of our menu from the original test program, and just modified it to test the hash table instead of the linked list. The modifications were pretty straightforward and didn't require too much alteration to have the Part class mesh with our hashtable.

4. **Include a discussion of why you think the results for Task 5 were what they were. This should suggest ideas for further investigation.**

The results for our Task 5 were very heavily in favor of chained hash tables. The efficiency difference was about 10x+ in favor of chained hash tables. Chained hash tables preserve the insertion order of elements, making them useful for applications that require elements to be processed in the order they were added. This is not possible with a normal hash table, which does not provide any guarantees about the order of elements. We hypothesized that this is one of the main reasons that the chained hash tables were so efficient in comparison to the normal hash tables. Another large benefit to using the chained hash tables is their resilience to collisions. Linked hash tables are more resilient to hash collisions than simple hash tables because they use a linked list to store multiple values that map to the same index. This makes it easier to handle collisions without sacrificing performance (no possibility of traversing the entire table to find a value that got linearly mapped several times.

Further investigation could include resizing behavior and memory usage. Both hash tables and chained hash tables may need to be resized as the number of elements in the table grows. Comparing the resizing behavior of hash tables to that of chained hash tables can help evaluate their flexibility in handling changes in the number of elements. Additionally, both hash tables and chained hash tables require memory to store their keys and values, but chained hash tables require additional memory to store the linked list nodes that are used to handle collisions. Comparing the memory usage of hash tables to that of chained hash tables can help evaluate their space efficiency and could be an interesting test condition in a future lab.