**1.**

The concepts explored in this lab include hashing algorithms and linked list vs array performance in regard to hash tables. Hashing algorithms are a critically important topic to study not just for this course, but in careers as well. In the context of the course, hashing serves as a nice way to organize data in a simple manner, ensuring that the data we receive in a table is exactly what we want. In the context of the professional world, hashing is a key component of security. Any time you are entering a password into a site, that password is put through a hashing algorithm and stored into a database. The stronger the hashing algorithm, the more secure your users’ information is. As far as the linked list vs array performance goes, this lab aims to help show when one might be better than another. It is important in a professional environment to understand when to use which one. With this knowledge, you can maximize efficiency for that particular task. In this case, using a chained hash table that implements linked list items allows for a bit more complicated code, but creates a freely expanding bucket for each hash location. It requires less total operations when adding and getting items but could require more memory and code. A regular hash table that simply uses arrays allows for clear and concise code but is limited in size and can promote clustering.

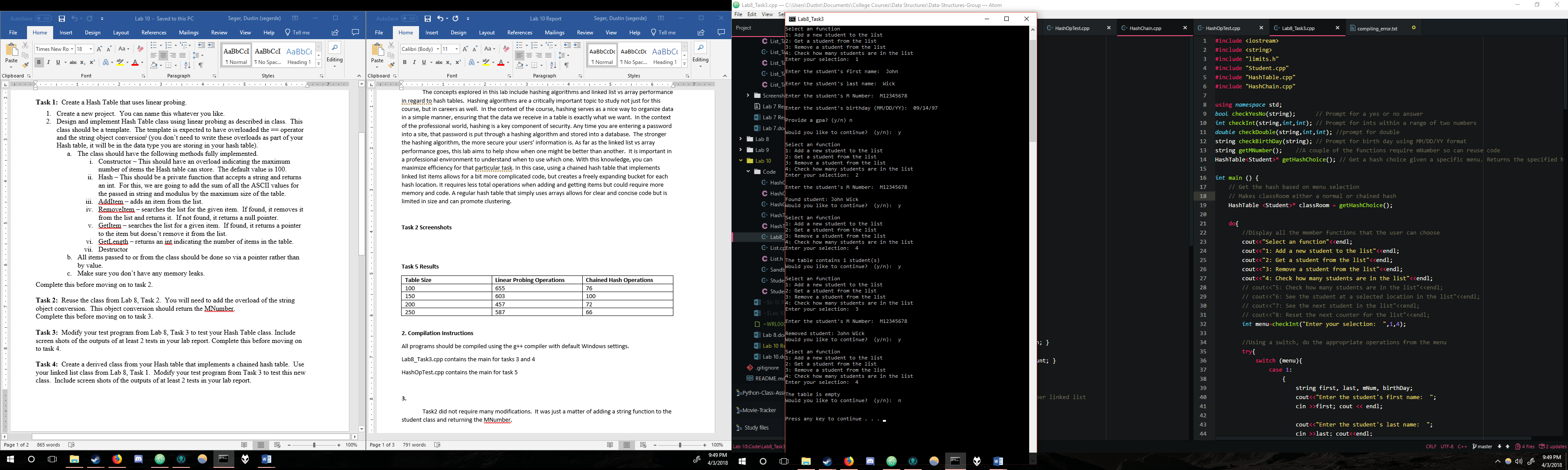


Figure - Task 3 add, get, remove, and length tests

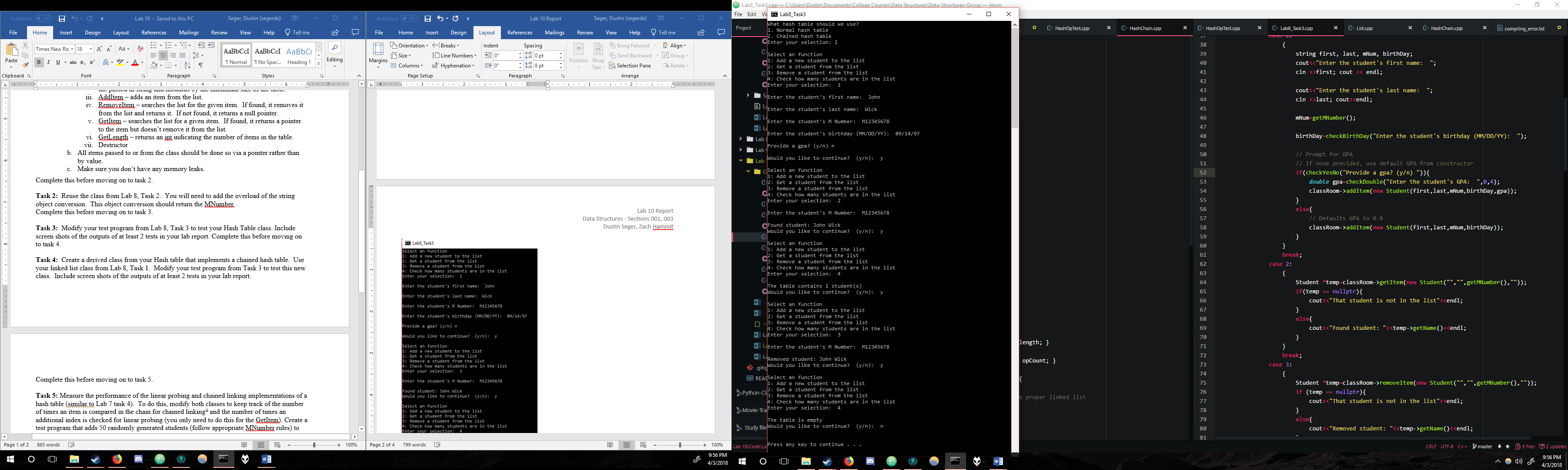


Figure - Task 4 add, get, remove, and length tests

**Task 5 Results**

|  |  |  |
| --- | --- | --- |
| **Table Size** | **Linear Probing Operations** | **Chained Hash Operations** |
| 100 | 655 | 76 |
| 150 | 603 | 100 |
| 200 | 457 | 72 |
| 250 | 587 | 66 |

**2. Compilation Instructions**

All programs should be compiled using the g++ compiler with default Windows settings.

Lab8\_Task3.cpp contains the main for tasks 3 and 4

HashOpTest.cpp contains the main for task 5

**3.**

Task2 did not require many modifications. It was just a matter of adding a string function to the student class and returning the MNumber.

Implementing Task 3 was a somewhat straightforward process. It was only a matter of changing out the student list that was originally used in Lab8Task3.cpp and making it use our custom Hash Table.

To implement Task 4, a function was added to Lab8Task3.cpp that asks the user if they would like to use our new chained hash Table or a linear probing hash table. Most of the functions required overriding to use a table that is an array of lists rather than an array of custom data class values. However, since the list logic was already implemented, we simply needed to use the functions that were in place from our other files. Choosing either array in the test file should produce the same results, they just work differently in theory.

**4.**

The hash table sees a great increase in performance as the size of the table is made bigger. This would appear to be due to the lower number of collisions as the table grows in size. This makes sense as there are many possible MNumbers, but only maxSize of table possible hashes. A test of a table size of 50 items should yield a massive operation count. As a table of this size would start to fill up, it would potentially have to probe through tens of spots before it found a proper place. This also should be tested with some table size around the max Mnumber. In such a situation, due to the hashing algorithm, there would be very few if any collisions.

The hash chain table is a lot more resistant to change in the table size than the normal hash table. This comes down to collisions not mattering as much for this type of hash table. This is due to its nature of not really having to waste operations dealing with collisions, rather just adding the data onto the end of a linked list. A way to evaluate this further would be to do the same tests proposed in the Hash Table Discussion. In a table size of 50, I would expect the linearly probed hash table to have massively higher operations performed than the chained hash table. In a test with max MNumber sized tables, the two types of hash tables would be expected to have similar performance.

Although the data itself varied due to the random MNumber values produced, we can see that the trends are heading downward. This variance also occurred due to the MNumber’s ASCII values only typically being within a certain range, promoting clustering within a set of indexes. A more accurate representation of the tables’ performances could be produced with a better hash function and/or more trials for each array size.

**Contributions**

Zach created Hash Chain and Table  
Dustin created the main and edited Hash Chain and Table  
The work is equitable.