

Lab 10 Report

Objectives in the Lab:

This lab explores the ability to use Hashing Tables as a data structure, either in the 1D form with linear probing or the 2D with a Linked List which is able to hold many of the hashes. These are useful as it is a versatile structure which is easily able to be implemented. This is then useful in the CS Field as it makes sure that the data is stored in a quick manner which is also able to be created easily due to the simple data structure.

Task 3 Screenshot:

```
1. Add Item to the Hash Table
2. Remove Item from the List
3. Check if Item is in the List
4. Gets the size of the List
5. Reset the List

Operation to Perform: 0
[537, 480, 272, 578, 52, 439, 588, 482, 522, 963, 528, 776, 133, 882, 821, 478, 943, 612, 912, 180]

Added 20 Random Items

Run again?(y/n): y
1. Add Item to the Hash Table
2. Remove Item from the List
3. Check if Item is in the List
4. Gets the size of the List
5. Reset the List

Operation to Perform: 2
Enter the SKU of the item to find: 52

Item was found and removed contents are:
Part Number: 52
Description: this is the item of sku 52
Price: $3

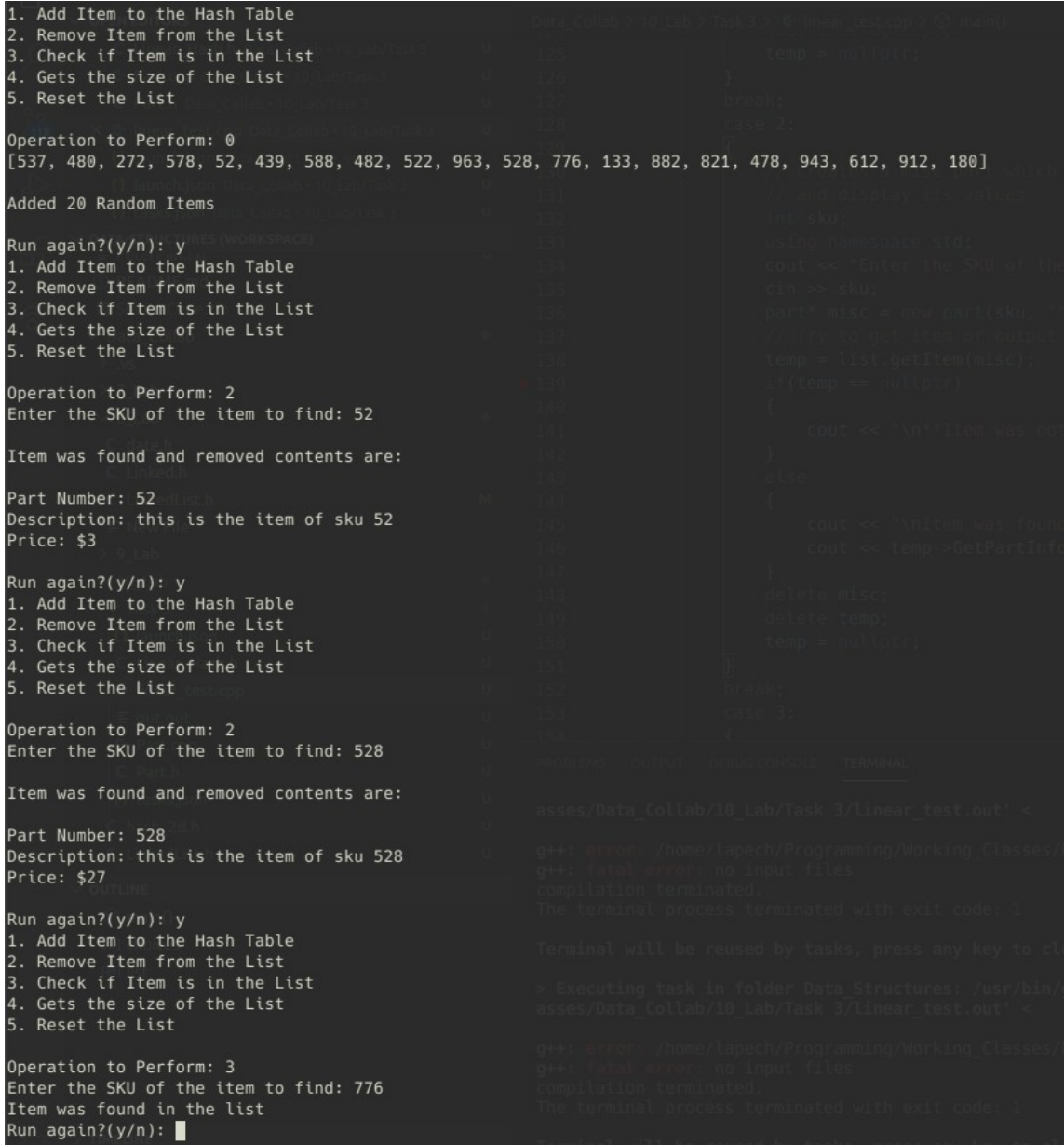
Run again?(y/n): y
1. Add Item to the Hash Table
2. Remove Item from the List
3. Check if Item is in the List
4. Gets the size of the List
5. Reset the List

Operation to Perform: 2
Enter the SKU of the item to find: 528

Item was found and removed contents are:
Part Number: 528
Description: this is the item of sku 528
Price: $27

Run again?(y/n): y
1. Add Item to the Hash Table
2. Remove Item from the List
3. Check if Item is in the List
4. Gets the size of the List
5. Reset the List

Operation to Perform: 3
Enter the SKU of the item to find: 776
Item was found in the list
Run again?(y/n):
```



Task 5 Chart:

Number of Items	1D Hash Table	2D Hash Table
50	1077	68
150	15813	396
200	28189	568
250	45781	758

Modifications from Task 2 to 3:

No major portions of code had to be changed as due to c++'s ability to implicitly convert a data type, this meant that the data type of part could automatically cast into a string using the overloaded string() conversion operator.

Results of Task 5:

I thought this was interesting, I had originally thought it would be the other way around but it makes sense now. The 1D Hash Table, was larger due to the need to constantly handle a large number of checks because of moving the table around. The hash would need to be moved around to ensure that no gaps formed which would have prevented the table from getting or inserting correct values. The 2D Table fixed this as each portion only had to deal with values that were of the same hash. This limited the number of collisions between 2 separate hashes attempting to overwrite each other and the issues that would come from that.

Something further to investigate would be if there could be some way to not use an array but a linked list which maybe had a dictionary type of data structure. This could possibly keep the normal way in which it all functions will limiting how much data is needed.

Contribution:

Chad:

- Created 1D Hash Table
- Created 2D Hash Table
- Created Tester
- Created Report

Colton:

- Got the 2D Hash Table working with the Linked List Class
- Ported the Part.h and overloaded the string() operator
- Did debugging on the 1D Hash Table Remove function

Compiling:

Important to use mingw-g++ if possible to run the code as this worked best for them.

The code is included in 2 separate folders which include task 3 related programs and task 5 related programs

The Code is also precompiled into 32x and 64x .exe for windows files and .out files for linux | ubuntu