C950 Task-2 WGUPS Write-Up

(Task-2: The implementation phase of the WGUPS Routing Program).

(Zip your source code and upload it with this file)

Donald Azevedo

ID #011371173

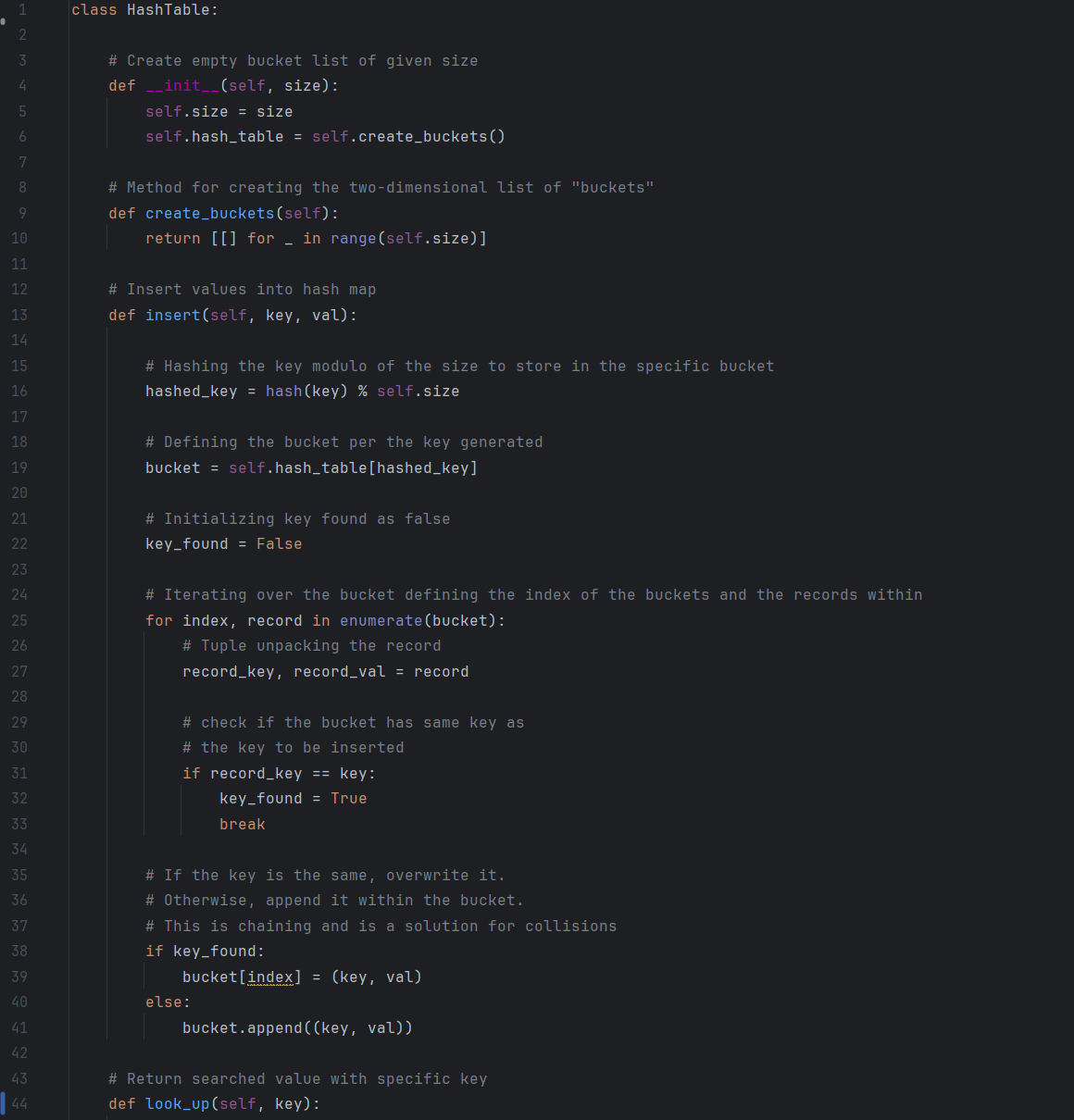
WGU Email: dazev15@wgu.edu

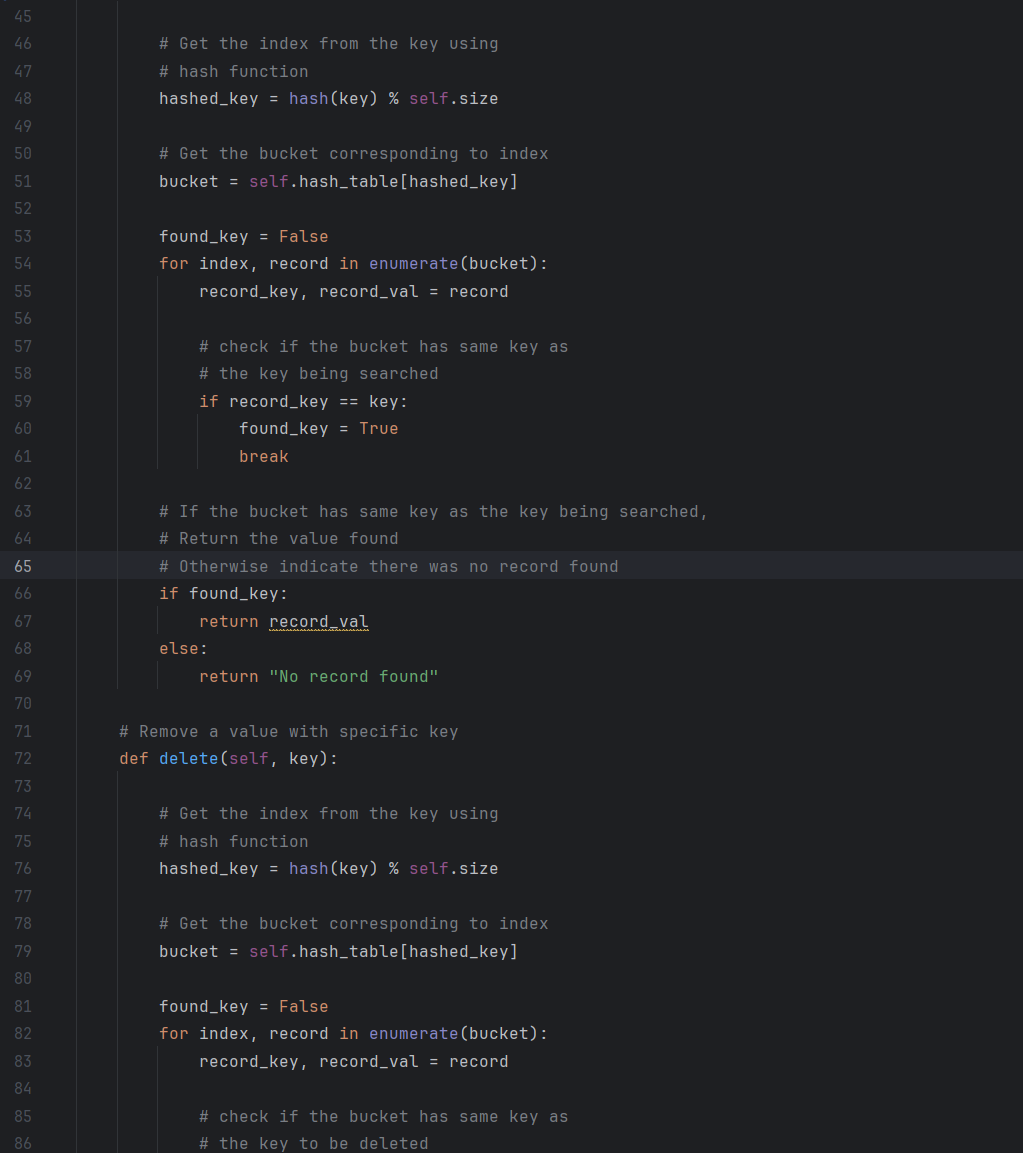
Date: 4/20/24

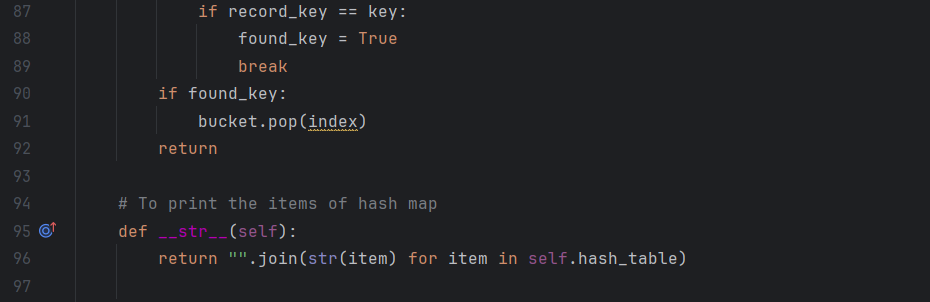
C950 Data Structures and Algorithms II

# A. Hash Table

See Hash table in the directory: Classes\HashTable.py

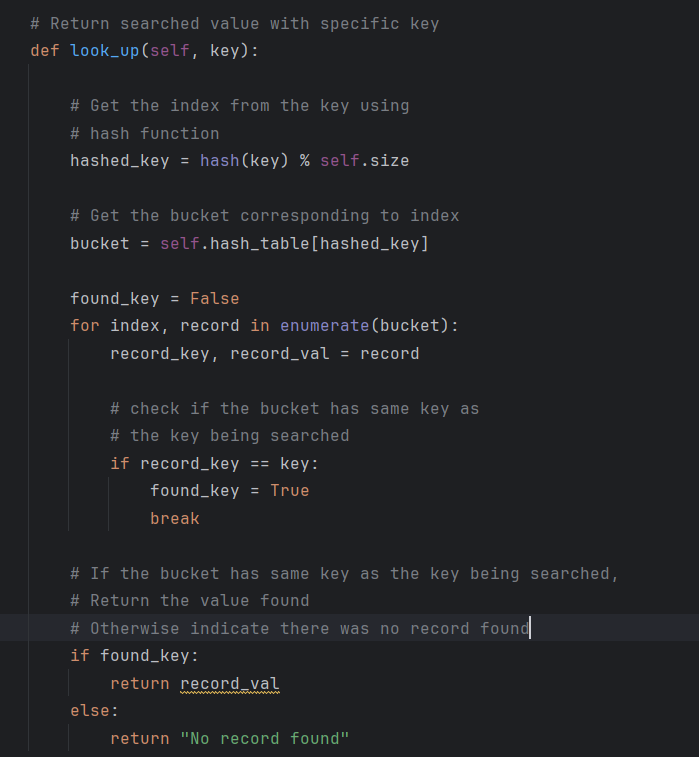
Full Hash Table Screen Shots:  




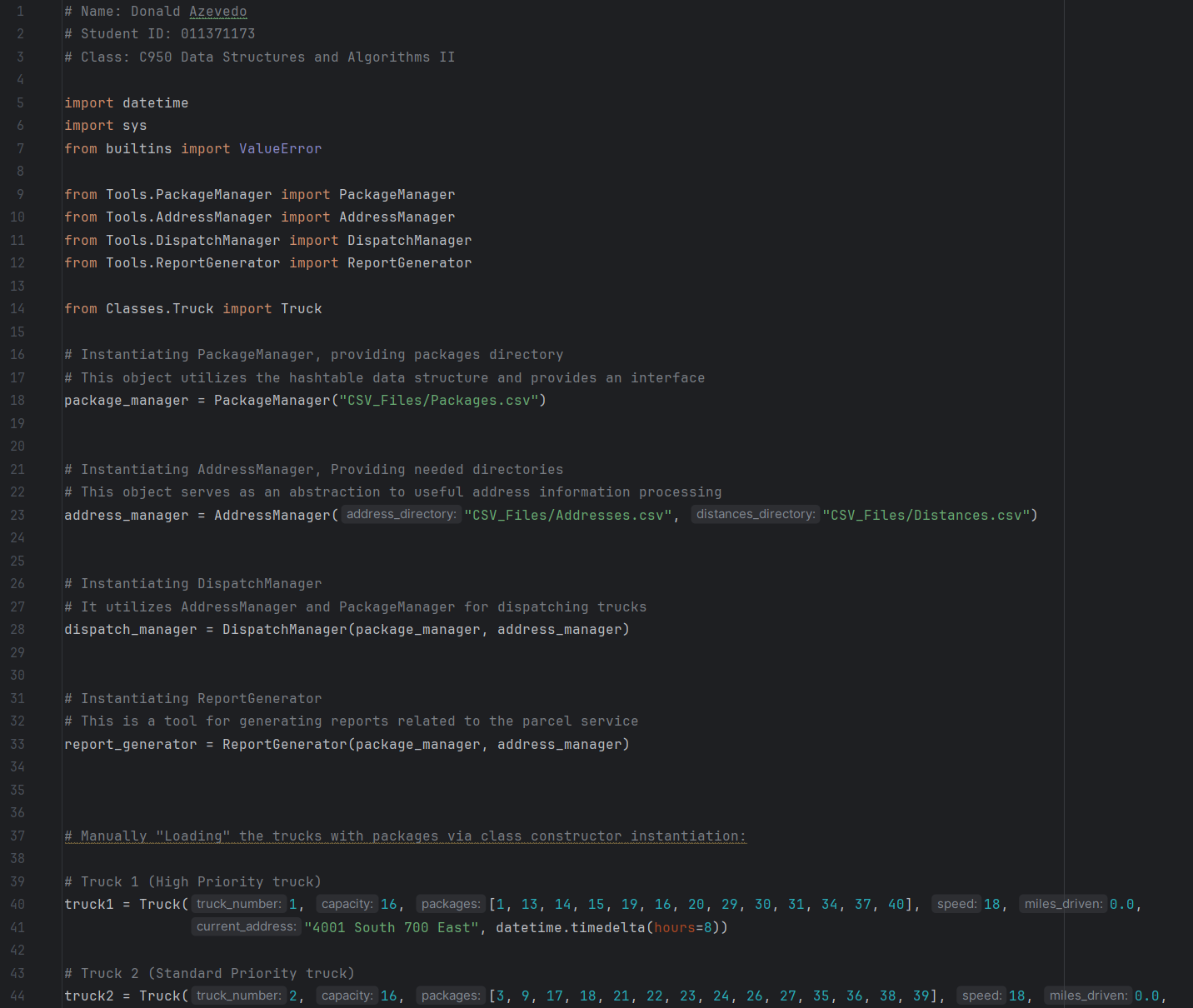


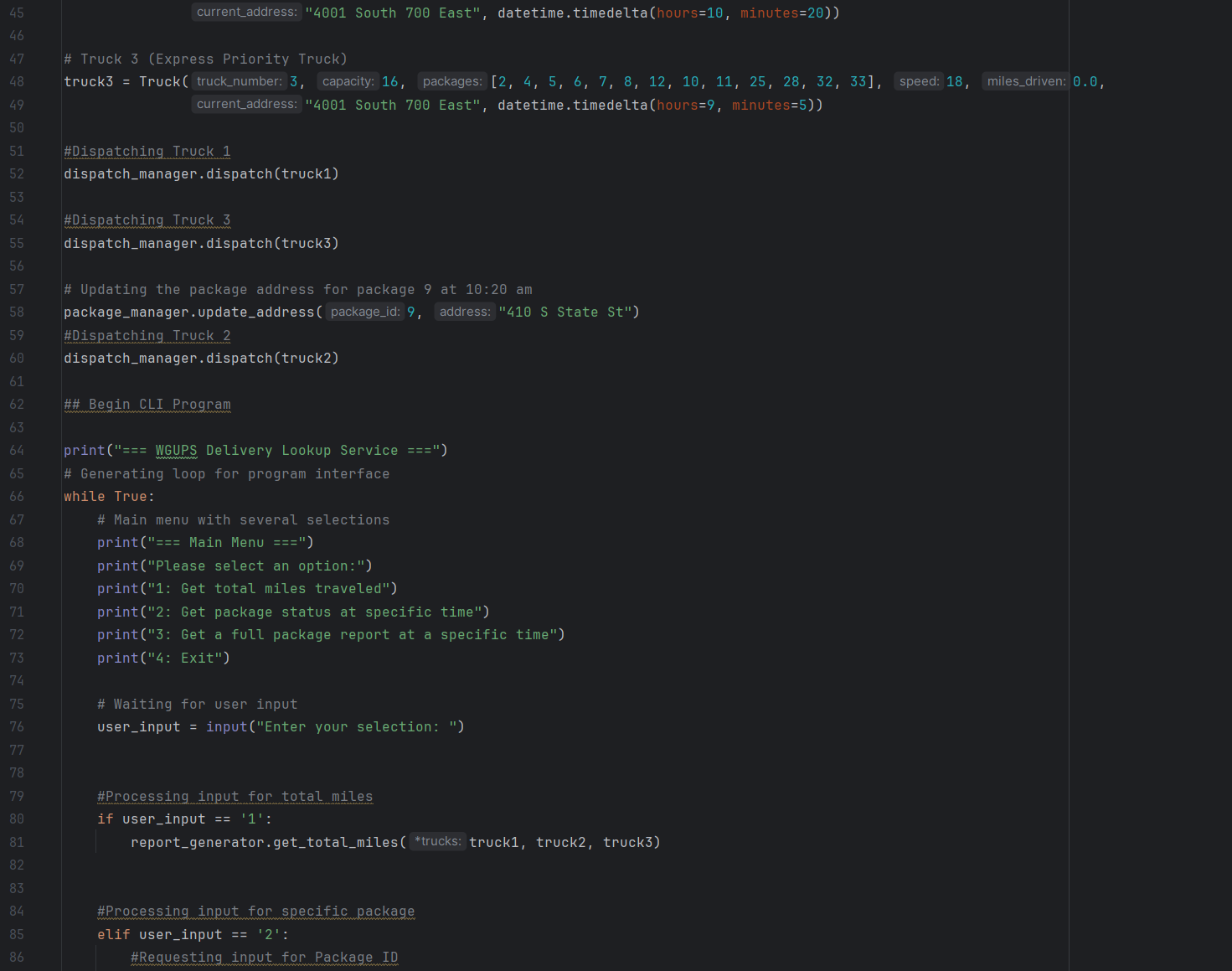
# B. Look-Up Functions

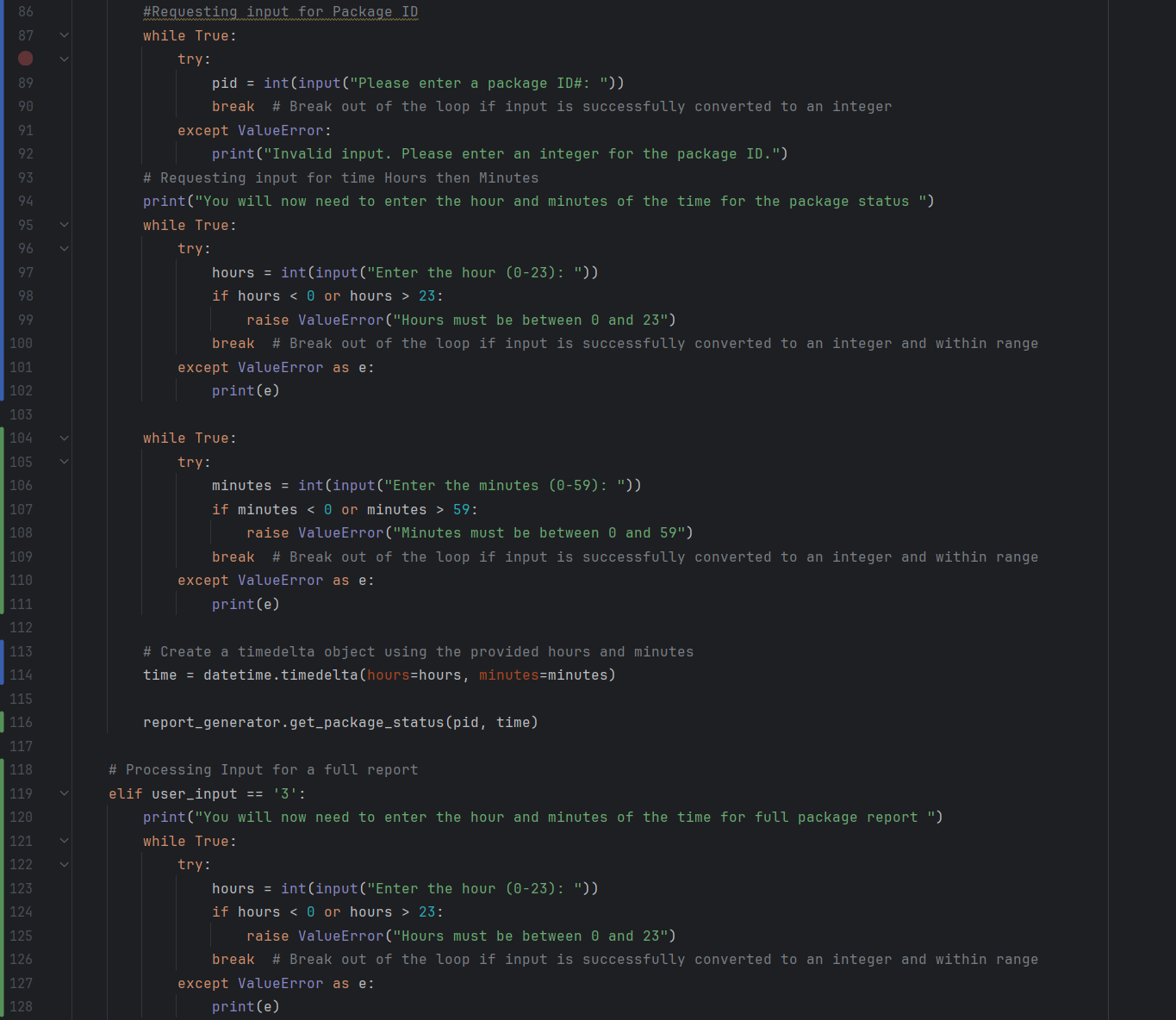
The Lookup function is located in the Classes\HashTable.py

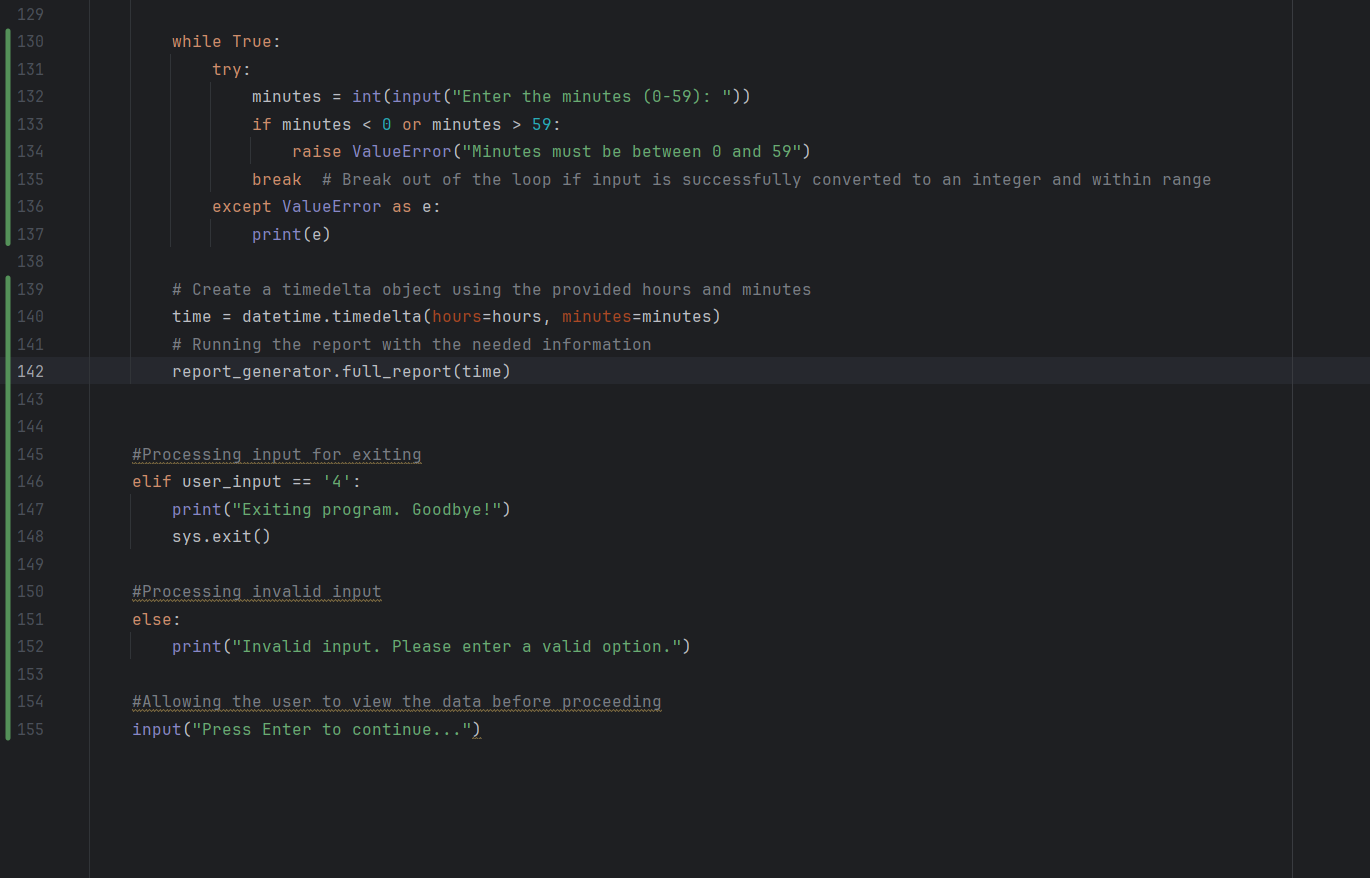
Here is a screenshot:  


# C. Original Code

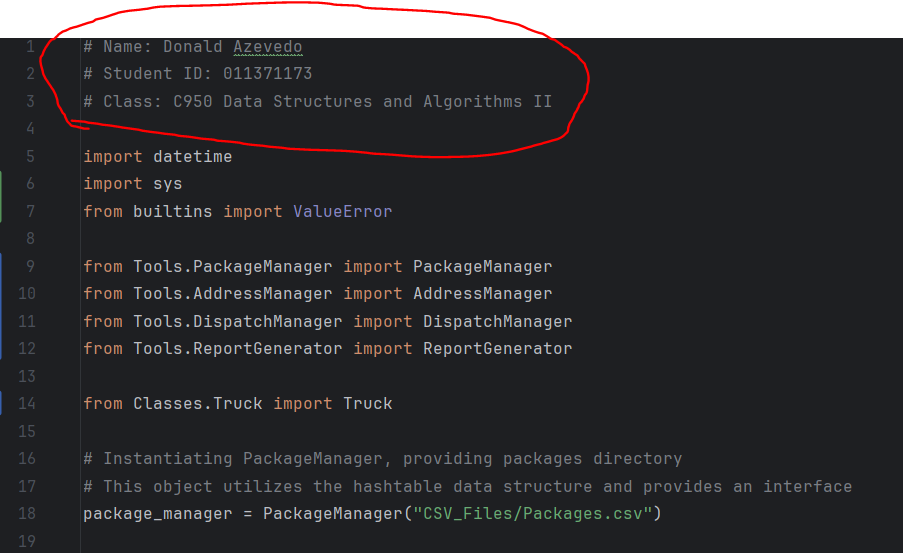
Main.py code:  




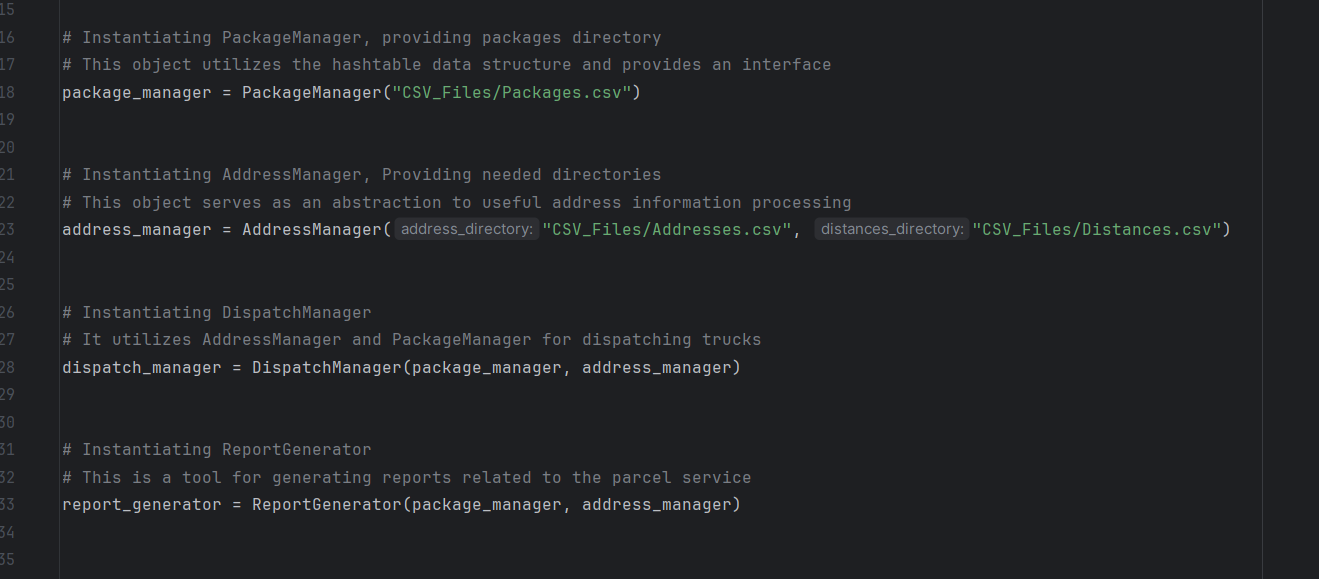




# C1. Identification Information

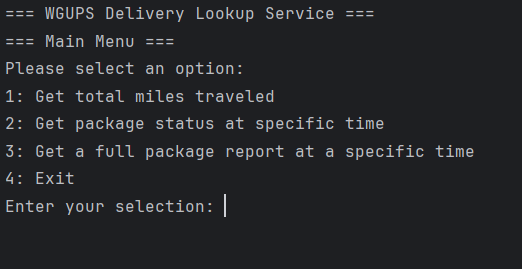


# C2. Process and Flow Comments

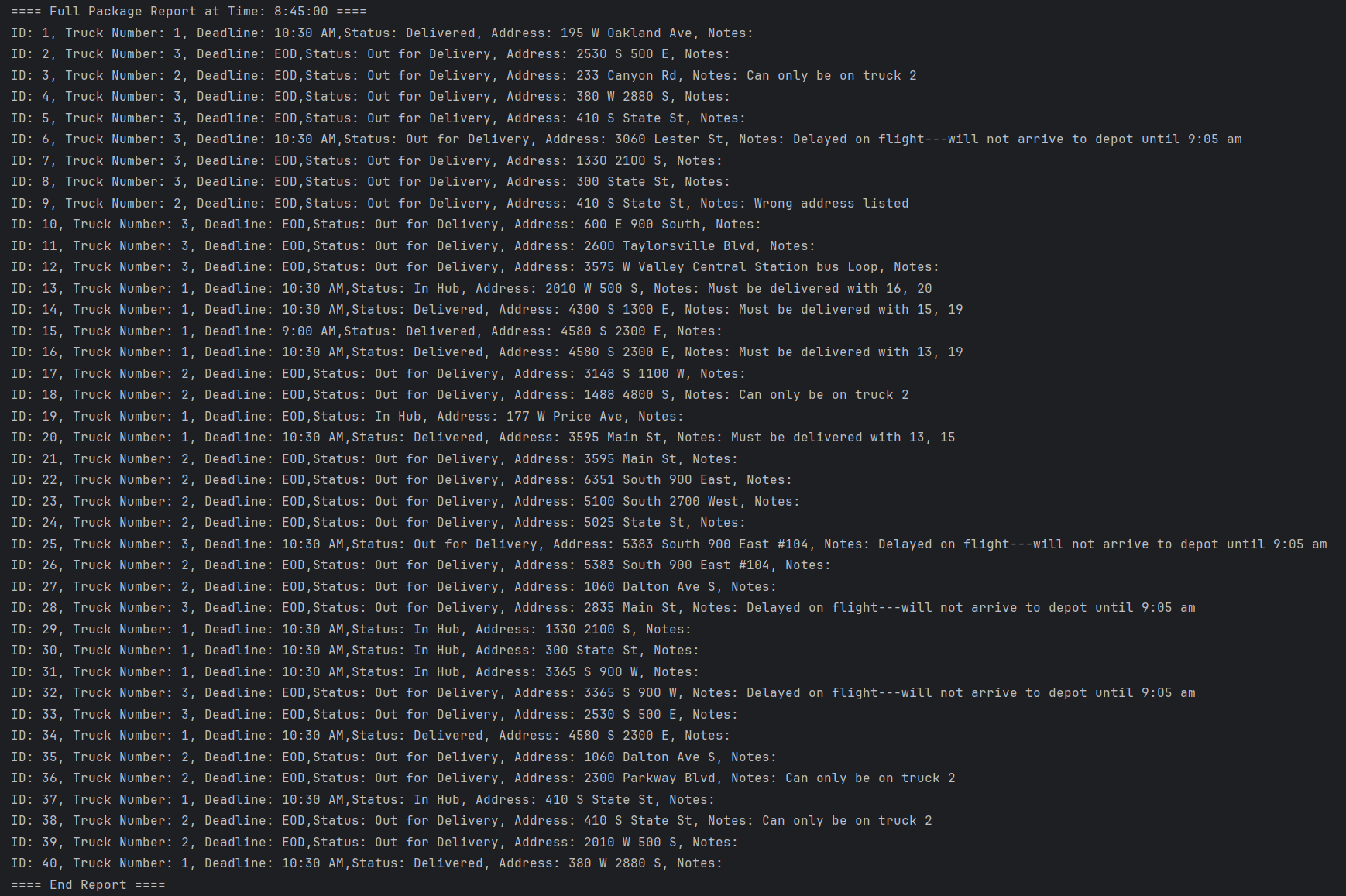


# D. Interface

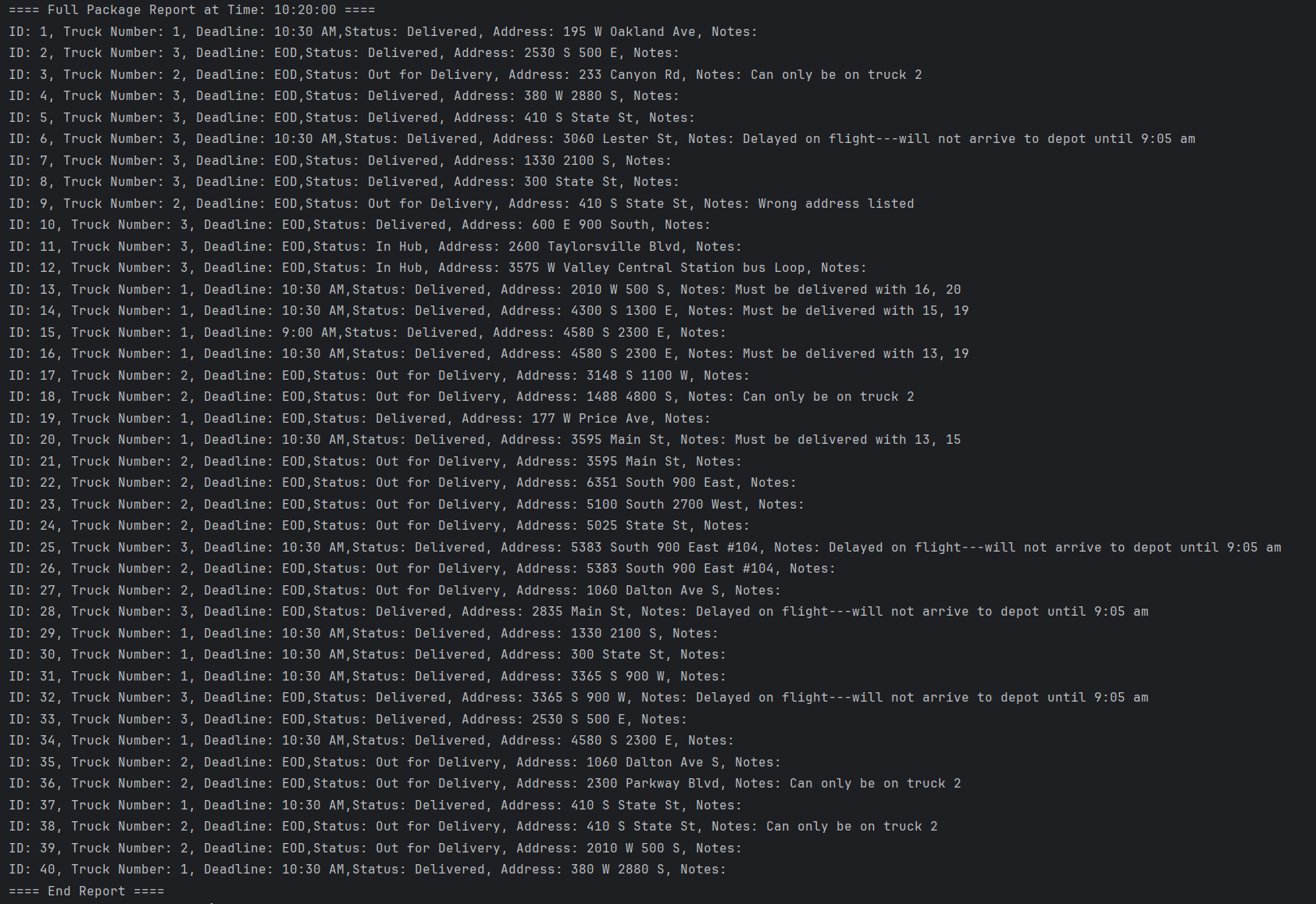
Initial Interface:



# D1. First Status Check

Full Status Report at 8:45 AM:  


# D2. Second Status Check

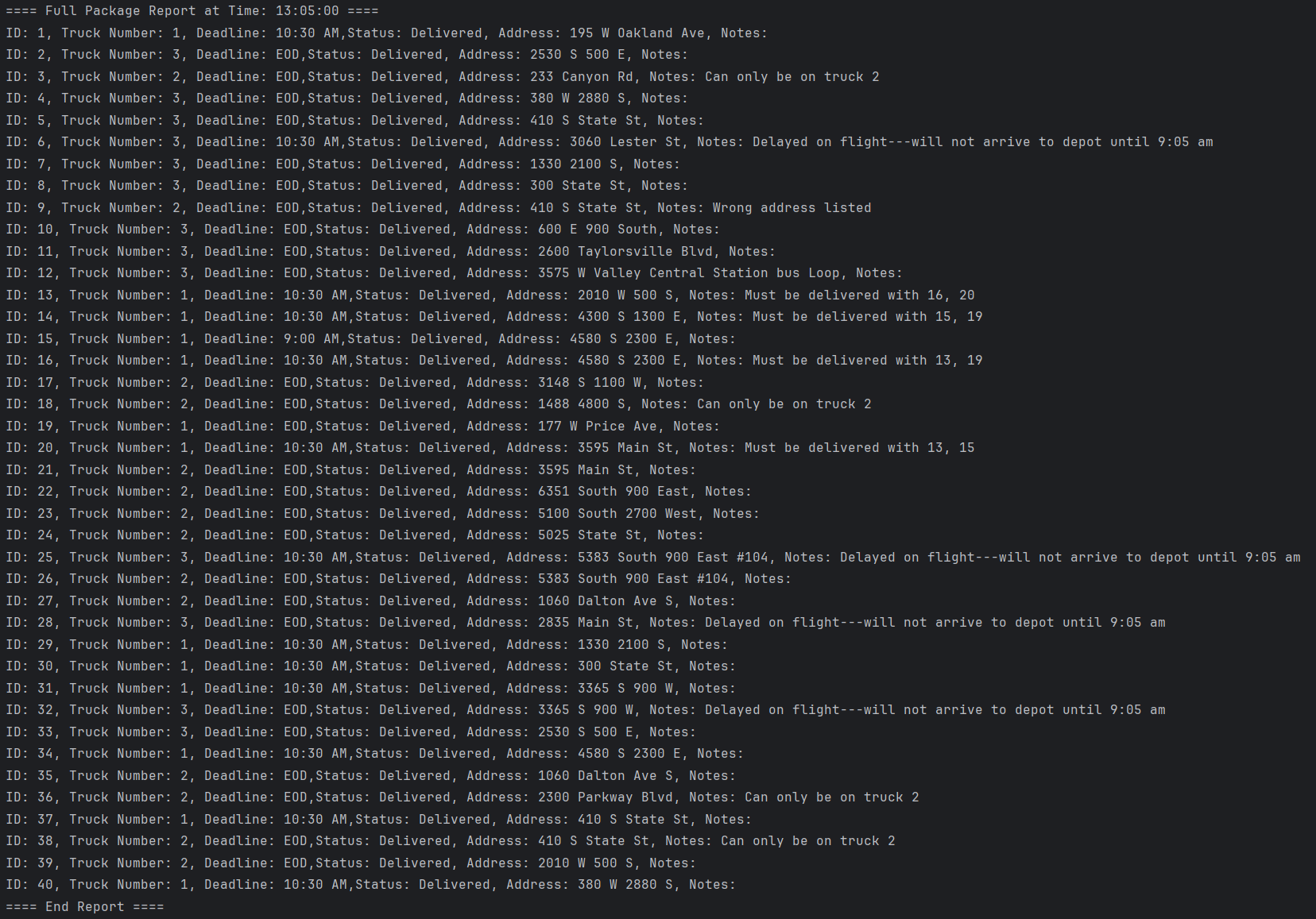
Full Status Report at 10:20 AM:  


# D3. Third Status Check

Full Status Report at 1:05 PM (13:05)

# E. Screenshot of Code Execution

Screenshot goes here

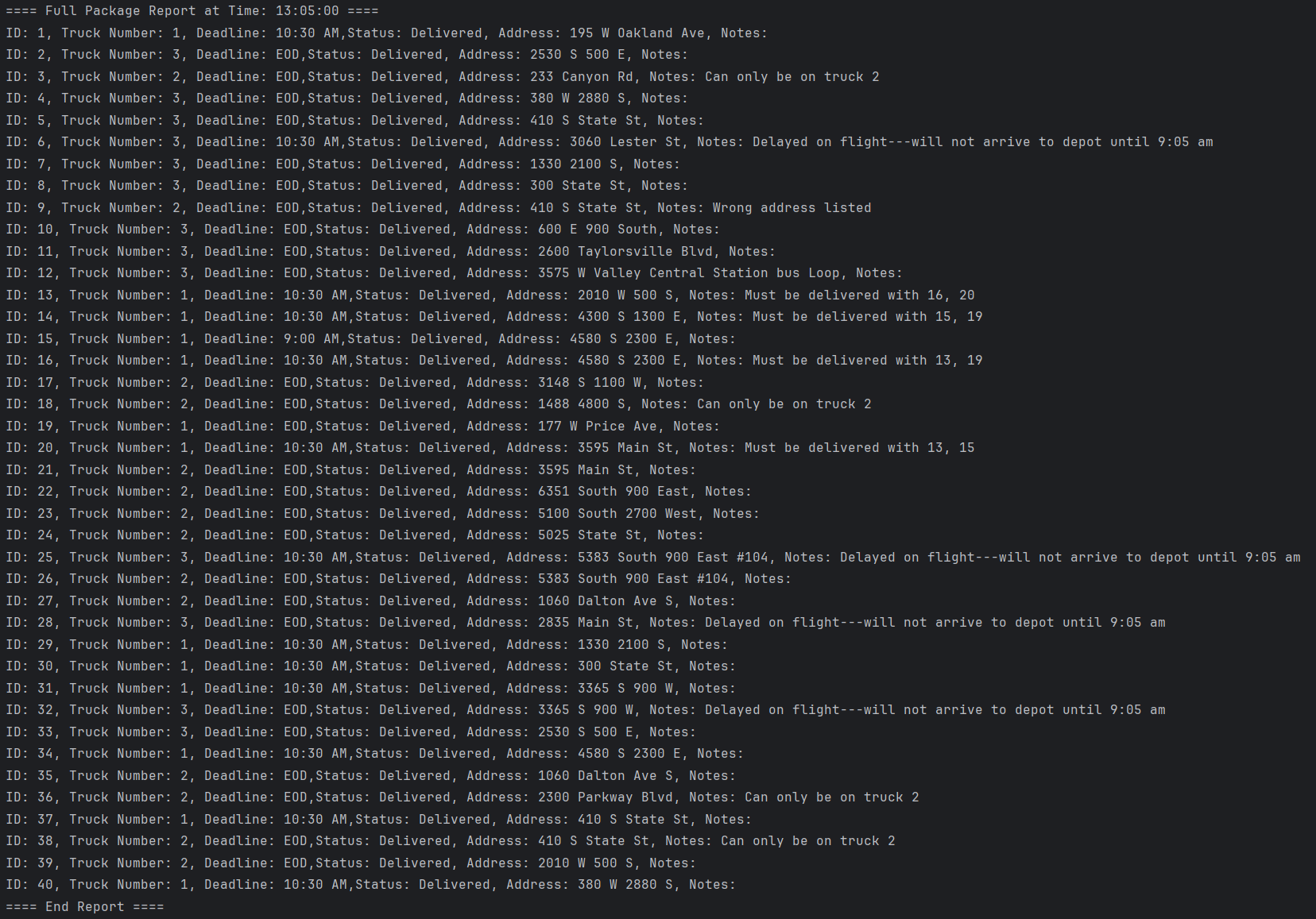


# F1. Strengths of the Chosen Algorithm

This approach has a few decent strengths. The delivery Algorithm uses the nearest neighbor method with a priority system. A benefit of the nearest neighbor method is it is not very computational intensive but still provides a reasonably efficient route.

Another strength to this approach is that in utilizing the priority system, it is dynamic and able to ensure higher priority packages “override” the nearest neighbor method and are put into their subset of nearest neighbors.

# F2. Verification of Algorithm

This screenshot from earlier displays the notes of each package. This confirmed that all the packages are on the correct trucks, met the deadlines, and are bundled with the correct packages when necessary.  


# F3. Other Possible Algorithms

Some other possible algorithms would be Random insertion and Brute Force.

# F3a. Algorithm Differences

The Random insertion algorithm is not computationally intensive. It also could meet the requirements of the project. However, it can be somewhat unreliable in situations where priorities exist within the packages.

The Brute Force method is probably the most computationally intensive. This algorithm has a time complexity of O(n!). However, this will find the absolute shortest route between edges.

The main problem with this algorithm is it does not scale well at all. For example, 10 packages would take about 3 million comparisons. Whereas, 14 packages would take approximately 87 billion comparisons. Even with modern computing hardware, this could become impossible to implement pretty quickly.

# G. Different Approach

If I were to work on this project again, I would establish a solid approach to automatically loading the trucks. I would categorize the packages into several subcategories. I would then categorize the trucks as well. I would use these categories to created a “weight” system for evenly distributing between the trucks. It would be important to take into account many factors such as capacity and deadlines.

# H. Verification of Data Structure

In looking at the algorithm, you can see that the data structure utilizes “buckets.” These buckets are associated with addresses in memory. The hashing method directly accesses this memory location. Within the memory location contains a list. This list exists to handle collisions. This process is known as chaining.

# H1. Other Data Structures

Two data structure that could accomplish this task would be a Singly-Linked List and a binary search tree.

# H1a. Data Structure Differences

The binary search tree can be quite an efficient searching algorithm. It does require the items to be in order. This allows you to narrow now the search by eliminating half of the elements every operation. The worst-case time complexity is O(log(n)) (Baeldung, 2023). Unfortunately, this algorithm is still not at efficient for time complexity as a hash table.

The singly-linked list is a very simple to implement algorithm. It is convenient in that it is dynamic. Also insertion and deletion time complexity is quite fast (O(1)). (Baeldung, 2023). However, search operations can be quite slower than a hash table that generally has a O(1) time complexity.

# I. Sources

Baeldung (2023, March). *Binary Trees vs. Linked Lists vs. Hash Tables*

Retrieved April 20, 2024 from <https://www.baeldung.com/cs/binary-trees-vs-linked-lists-vs-hash-tables>

# J. Professional Communication