Program #2 Review

Written by Alexander DuPree

Data Structure Performance

Program #2 required the use of an doubly linked list to represent a fleet of differing types of rental transport vehicles. Because the program required the client to examine each vehicle individually the doubly linked list was perfectly suited for insertion, removal, and iteration operations. We were able to allocate a dynamic number of rental transport objects at runtime and traverse the list both forward and backward modifying each object or removing them at the clients demand.

The largest drawback of this data structure, however, is the linear traversal and other shortcomings common to linked lists. In it's current implementation the client can only access each rental transport object linearly, and is unable to sort the list by any sort of criteria.

Object Oriented Design

The design of program #2 emphasized the use of inheritence and dynamic binding to utilize a single data structure that could manage a myriad of objects within a hieriarchy. This was accomplished by pushing up common methods up to a abstract base class and using base class pointers to call these varied functions. The Rental_Transport class acted as the abstract base class for this assignment and contained a pure virtual functions that the derived classes defined. This allowed our doubly linked list of base class pointers to utilize the same function call to invoke a variety of derived dependent functions. This inheritence structure helped reduce code bloat by reusing the relevant parts of code that applied to all classes within the hierarchy.

GNU Debugger Discussion

The GDB was an invaluable tool during the testing and production of program #1's code. I also utilized the catch2 unit testing framework to establish a suite of tests for my program. This test suite coupled with the GDB allowed me to catch and pinpoint errors in my code almost immediately. If a certain test failed after a lengthy refactor I could fire up GDB, set a break point at the test and step through each line of could to identify the error. The debugger, coupled with unit tests, was able to exponentially increase my productivity as a programmer as I spent more time writing code and less time debugging. The GDB was also important in affirming that my program was utilizing dynamic binding correctly, as I could step through each line of code through the debugger and ensure the correct function was being called from the hieriarchy.