Kyle Zalewski

CS202

11/2/2018

# Programming Project 2 Design Analysis

This project was a decent challenge for me. I learned quite a lot about inheritance relationships as well as containing relationships, and the various hoops to jump when moving from struct-based to class-based nodes. One of these hoops was passing functions and variables around between members of various classes. This posed a challenge when troubleshooting, since an issue whose source would have been obvious in simpler code became a bit more difficult to find when functions call other functions several times over.

The challenge was worth the effort since my project ended up more properly object-oriented as a result. My int main was very short compared to past projects, and I was able to clearly see the advantage of single-line functions providing the client-side functionality to my program. The backend source code grew rapidly in length, however.

In future projects, I will challenge myself to logically break up source and header files into smaller and more sensible separate files. I was hesitant to do so in this project because I ran into some issues with undeclared data members when class declarations were out of order. I chose not to separate the source code to avoid bringing this issue back (and having to modify include statements to compensate).

I believe I now have a decent grasp on the idea of abstract base classes. In my case, the abstract base class was my transportation class, which I ensured by including a pure virtual display function. I implemented this function in each of my derived classes and was therefore able to set up my output based on the type of object being displayed. I ran into an interesting issue where deletion led to derived class objects being unable to display, and it was somewhat unclear why. It appeared that the derived class object was somehow being converted to a base class object (which obviously can’t exist).

I implemented copy constructors for the two classes that managed dynamic memory, which were the base transportation class and the derived Zipcar class. These classes had dynamic character arrays, which I copied to new memory in the copy constructors for those classes. This ensured deep copy and guaranteed that I wouldn’t lose data in the event of a copy.

In future projects, I would experiment with more RTTI and look to properly implement initialization lists. I believe that some of my mystery issues may have been caused by improper initialization.