Kyle Zalewski

CS202

Object Oriented Design

For this project, I’ve decided to tackle the comparatively more complex red-black tree structure. This leaves the less complicated binary search tree structure for project four, where I will be working with a language with which I’m not as familiar. For the data structure itself, I’ll most likely be using the tried and true containing relationship between the Tree and Node classes. Since I’m familiar with this method, it will allow me to emphasize the operator overloading portion of the project, rather than spend too much time on the data structure portion. The Node class will contain the base class for the apps in the project.

My base class will be Apps, and will have derived classes Weather, News, and Game. The three derived classes will represent the differences between those apps, and commonalities will be pushed up to the base class. My base class will be abstract, because for the purposes of this assignment, there will be no need to create an object that is simply an app. This class will handle all search functions, unless I find a need later on to include more specific search functions that are unique to one or more of the derived classes.

Setters and getters will be needed in the Node class, but will be avoided elsewhere by careful consideration into the jobs of each other class. For organizing the data in the structure, the setters and getters in the Node class are unfortunately unavoidable. If I end up using a design that doesn’t use a Node class, the setters and getters will be avoidable. I’m considering this option as well.

There are several unique traits to the derived classes. The Game class will have a data member showing the high score on the app. The News app will show the top headline. The Weather app will have a short description of the current local weather, and a current temperature.

The biggest challenge will probably be implementing the data structure. I’ll have to refresh my memory on the red-black tree and implement that first so that I can spend the remaining time implementing the large number of operator overloads that will be required for this project. The red-black tree has some very distinct advantages for search efficiency, so it will be a good topic to know, as there is very little extra memory overhead versus a standard binary search tree. Though it is more complex than the binary search tree, I believe there is a distinct advantage here since the project is very search-heavy.

For operator overloads, I will require the full suite of operators and will likely implement them in every class in the hierarchy. If I continue to use the Node class, there are several operator overloads that will assist in using that class as well. The assignment and arithmetic assignment operators will be very useful in creating a client-side environment that allows predictable behavior from my classes. If I can properly implement these overloads in addition to the insertion and extraction operators, my classes will behave very similarly to how they would if they were built into the language. There may also be a way I can tie in the overload of the insertion/extraction operators into the requirement for this program to support the use of an external file. I’ll continue to investigate whether or not this is an approach that supports predictable behavior of my class objects. Once the operator overloads are complete, the rest of the code can potentially become much shorter and more readable. This will make the rest of the coding process easier, so I intend to take care of the operator overloading before tackling other portions of the assignment.