Kyle Zalewski

CS 202 Project 3

Design Analysis

Design wise, this program appeared from the beginning to be significantly less complex than the last. In practice, more and more complexity seemed to be required as I made my way through coding. My main goal was to avoid getters and setters as much as possible, which placed a special burden on operator overloading.

Operator overloads were necessary for any class managing dynamic memory. For my design, these classes were my abstract base class food, the derived class mainCourse, and the node and tree classes. All of these classes required an overloaded assignment operator. I decided to use this for changing the name of a class object, so that it would behave in a somewhat expected manner and make changes to the dynamic members in the class.

I used overloaded relational and equality operators to sort insertion of new food pointers into the tree. This seemed like the most sensible way to use the operators, since comparing Boolean values numerically wasn’t necessary, and didn’t really make sense for the application (though it would have been possible using operator overloading).

Another new concept that I implemented in this program was RTTI. I found out that when displaying objects, this was the best way to ensure that the proper display functions and overloads were being called through the hierarchy. From main, there were several function calls and operator overloads (extraction operator) between the client program and the actual display of information, so ensuring that the proper type of object pointers were being passed through proved to be essential. This ensured that when I went to display the information, the proper dereferenced object data was displayed, rather than a nonsense memory address.

The most difficult task was implementing the data structure. Though I believe I have a decent grasp on how 2-3-4 trees work, the actual implementation of the algorithms is pretty difficult. Insertion was different from other data structures, because multiple class objects can move in and out of the same node. To solve this issue, insertion responsibility needed to be split between my tree and node classes. Once the division of tasks was decided, the rest of the coding was mostly chasing down syntactic errors. The decisions pertaining to the task division required the most thought, because I wanted to use as few single-use functions (getters/setters) as possible. To that end, it was crucial that the responsibilities each class had pertained to the data to which they had direct access.