Kyle Zalewski

CS202

Project 3 Design

For project three, I will be designing a holiday food inventory using a binary search tree data structure. I am more familiar with the BST structure than the optional 2-3-4 balanced tree, so I have chosen this structure until we have had time to review the 2-3-4 balanced tree in class. I will brush up on the concept and implementation of a 2-3-4 balanced tree for the next assignment, which will require that data structure alongside a switch to Java. I'm under the impression that Java has some memory management features that will make that implementation easier than in C++, but that remains to be confirmed.

In this project, I will have a class "tree" which will be responsible for organizing the nodes (containing relationship) of the binary search tree. Addition and removal of nodes will be functions in this class, called directly on a tree object from main. The contained class will be called "node" and will have the responsibility of setting up the pointers to children of the roots. Each node will contain an object of type "food" which is the class that will top the hierarchy of actual food items in the structure. I intend to make this an abstract base class, as this program will not deal with items that are non-specific "food" rather than one of the child classes.

My child classes of "food" will be "appetizers," "beverages," and "mainCourse." The beverage class will have the unique traits of "alcoholic" and "caffeinated," with some member functions to deal with the implications of those traits. Objects of the "appetizer" class have the unique trait of the possibility to be treated as a main course item, provided they are substantial enough. This class will have some member functions used to determine if an item fits that requirement. The "mainCourse" class will contain items that are specifically for serving at dinner time. There will be a dynamic character array used to catalog the type of meat used in the course, to exercise dynamic memory chops as well as give the user a better idea of what the main course consists of.

The "mainCourse" and "food" classes both contain character arrays that are dynamically allocated. As such, they will require destructors, copy constructors, and assignment operator overloads. The same is true for the "tree" and "node" classes, as they manage pointers for the binary search tree data structure.

The "food" class is the abstract base class, and thus will include a pure virtual display function. Accordingly, the child classes will need to include initialization lists to ensure that the copy constructor of the abstract base class is called when appropriate. I will upcast objects of the child classes so that the base class constructor is invoked upon creation, as well as their own constructor. This will allow all data to be set properly upon object creation and reduce or eliminate the need for any other type of "setter" or "getter" function.

I aim to further minimize the use of "getter" functions by making sure that my classes are not so specialized as to only be providers of one data member. I believe my current hierarchy will allow that to succeed. I will pay attention to the number of data members in my abstract base class and make

changes to child classes accordingly if I decide that my base class is not quite specific enough. As it sits now, my base class has lots of bool data members that represent various allergens in food. So far, I see these as needed considering lots of allergens can hide in unexpected foods and drinks.

I believe that the hierarchy I have chosen will allow me to keep my project object-oriented. The structure will allow me to have a very small client interface, and with proper operator overloading I will be able to treat my classes more like abstract data types and operate on them accordingly with very little code on the client side. My main goal is to minimize the overhead and responsibility of the client program by structuring my overloads to behave in ways that are similar to out-of-the-box C++ operators.