Jonathon Harris
CS202
Karla Fant
10/11/20

Relief Catalogue – Design Write Up

The purpose of this program is to create a catalogue of items donated from the community allowing others to search and take what they need. When someone searches for an item a history-linked list is creating which is in descending order of frequency of times it has been looked up. The relief items are in their own separate circular linked lists. The three relief items I chose to catalogue were food, clothing and shelter. The program should have at least five different classes with at least five functions for each of the classes. All list functionality should be done using recursion. Lastly, classes must be implemented using inheritance and one containment relationship.

For this assignment my class info is inherited by food, clothing, and shelter, which are inherited by their respective nodes: food node, clothing node, and shelter node. From there we have a List class that has three rear pointers: one for food, one for clothing and one for shelter – which acts as my containment relationship. The list class is responsible for maintaining and manipulating the three separate circular linked lists, because of that; I have a lot of duplicate functions. I have three add, remove, display, retrieve and remove all functions for food, shelter and clothing. We have yet to learn about dynamic binding so under the tools that I know currently this inefficient method is the best way I know. My history class is responsible for keeping a linked list of nodes that represent what the user has searched for. So history has add clothing, add food, add shelter all taking in on object of that class in

the argument. Initially I was going to have two containment relationships for my nodes and the list, but decided against it because of all the necessary wrapper functions. Because of the inheritance relationship my program will have a minimal amount of getters and setters. In fact, my nodes are the only classes with getters and setters these get and set the next pointers. This design makes the most sense to me because the info, relief items and nodes all act as a container for data. So really only the history and list classes are doing all the work. So it minimizes the interaction between functions, this creates a crystal clear guide into which classes need to do what. Overall I'm happy with the design of this program, as it's the first time I have programmed using inheritance. Certainly the inefficiency of writing three functions for each relief method is not lost on me. My add node functions have O(1) efficiency, the remove functions has O(n) efficiency, the retrieve function has O(n) efficiency, the search function has O(n) efficiency, and finally the remove all has O(n) efficiency. If the data structure were an array of linked lists or a dynamic array these functions would have a more efficient run time.

Inheritance allows for control to the members of a class, which is great in some circumstances, but the downside is it makes the code less malleable for future changes. So while a containment relationship may mean adding more wrapper functions thus causing our program to be larger. It makes it easier for future edits because of the lack of dependencies. So if I were to do this program again in the real world, I would setup the structure of the program with an emphasis on containment so future changes could be done with ease. Also I would use dynamic binding, which would greatly reduce the need for duplicate functions.