Program #2
CS 202 Programming Systems

***** Make sure to read the Background Information first!**
It applies to all programming assignments this term***

****THIS IS NO DESIGN WRITEUP and NO UML DIAGRAM for Program #2 ****

Program #2 – Purpose

In our first programming assignment, we experimented with the notion of OOP and breaking the problem down into multiple classes building a relatively large system for racing drones through an obstacle course. The purpose of that assignment was to get to know how we can create classes that have different relationships and experience hierarchies. In program #2 our focus shifts to getting experience developing a hierarchy that can be used with dynamic binding – so that the application program can use one line of code to call one of many functions. To get the full benefit of dynamic binding, we will look at a smaller problem with a predefined design.

Program #2 – General Information

Do you use Amazon as much as I do? How much do you shop at brick and mortar stores? Do you try out different computers at Frys and end up purchasing it from Amazon? Over the holidays, the delivery trucks were constantly driving up and down our neighborhood.

When we think about Amazon deliveries, delivery trucks (or vans) follow a particular route. They begin and end at the same place each day. They pick up deliveries at the warehouse and then deliver packages along the route. Some packages require signatures and can't be delivered without that; if no one is available to sign, they may have to revisit this later in the day – changing their route. Sometimes the delivery trucks may also pick up packages at specified locations.

Of course, Amazon is now expanding to food pickup and deliver along with packages! What will be next?

Program #2 – Building a Hierarchy using Dynamic Binding

For your second program, you will be creating a C++ OOP solution to support at least four different types of deliveries that can take place (i.e., deliveries, insured delivery (requiring a signature), package pickup, etc.)

The purpose of this assignment is to use and experience dynamic binding. The requirement for this application is to have at least **four DIFFERENT forms of deliveries supported**, derived from **a common abstract base class**! To use dynamic binding, there needs to be a self-similar interface among the derived class methods. In this case, for all types of

deliveries that you support, the user would like to receive a package, send a package, and so on. In the real world, there will be some differences as well, although there shouldn't be too many. For example, for packages that require signatures, we would need to be able to sign for a package. **Make sure to find at least one method that is different so that you can experience how to resolve such differences with RTTI. Everyone MUST use RTTI in this assignment at least once!**

It is expected that each type of delivery or pickup will require dynamic memory for items such as:

1. Name and Address of the sender
2. Name and Address of the receiver
3. Contents in the package

Program #2 – Data Structure Requirements

With program #2, our data structure should be of POINTERS to the ABSTRACT base class and then with upcasting cause each node to point to the appropriate type of item being delivered or picked-up. Implementation of the required data structure(s) requires full support of insert, removal, display, retrieval, and remove-all. Efficiency must be part of your data structure design. All traversal algorithms must be done with recursion (not loops!).

You will need to implement a circular linked list to represent the route of deliveries where each node will have the location for delivery or pickup and an abstract base class pointer pointing to the different objects that represent the type of deliveries taking place. For example, the route may contain a pick-up at the beginning at the warehouse location and then a delivery at PSU's CS department of a new DELL computer, a pickup of potstickers at the Phat Cart in Portland, and a delivery of those potstickers to 42 Highlander Street in Beaverton. Their last stop will be back at the warehouse at the end of the day.

Program #2 – Important C++ Syntax

Remember to support the following constructs as necessary:

1. Every class should have a default constructor
2. Every class that manages dynamic memory must have a destructor
3. Every class that manages dynamic memory must have a copy constructor
4. If a derived class has a copy constructor, then it MUST have an initialization list to cause the base class copy constructor to be invoked
5. Make sure to specify the abstract base class' destructor as virtual

IMPORTANT: ***OOP and dynamic binding are THE PRIMARY GOALS of this assignment!***