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 a delivery program. The purpose of that assignment was to get to know how we can create classes that have different relationships. 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

- When you take a course at PSU, each course has a set of different materials that you are responsible for reading or working through. There may be reading materials (e.g., books, power point slides with book reference names, chapter and page numbers), videos to watch (e.g., youtube with a URL) and exercises to work through (e.g., a list of problems).

Program #2 – Building a Hierarchy using Dynamic Binding

For your second program, you will be creating a C++ OOP solution to support at least three different types of course material. Create a base class for the general course material. This would be an abstract base class since we would never have general material without a specific assignment.

Then, create three different types of course material that will be derived from the general course materials. Implement the following two choices and then ADD ONE OTHER of your own choice. Make your type of material similar to the other two but yet different!

- For **reading** material, you would need the textbook title, author's name, chapter and page number assignment
- For **exercises**, create a list of exercises using a circular linked list. Every exercise should have a name and then a question (e.g., remove all nodes from a CLL recursively). A CLL is the ideal data structure so that the "rear" pointer points to the next question that hasn't been completed! It should change each time a question has been marked as being done.

The purpose of this assignment is to use and experience dynamic binding. The requirement for this application is to have at least **three DIFFERENT types of classes**, 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 course material, the user would like to add new material, remove material, display all review material, mark an assignment as completed. In the real world, there will be some differences as well, although there shouldn't be too many. **Make sure to find at least one method that is different so that you can experience how to resolve such differences.**

Program #2 – Data Structure Requirements

With program #2, we need a Doubly Linked List of base class pointers. Then using upcasting, each node can point to the appropriate type of course material. Implementation of the required data structure(s) requires full support of insert, removal, display, retrieval, and remove-all. This DLL allows a client to build a list for a PSU course that they are taking of all of the materials they need to progress through. Please support both a head and a tail pointer.

ALL repetitive algorithms should be implemented recursively to prepare for our midterm proficiency demos!

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!***