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. 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

- As we prepare for technical interviews (maybe you are thinking of looking at an internship opportunity after finishing with CS202), we find that different kinds of problems suit some data structures better than others. When preparing for one job, we might need to focus on parallel programming concepts and for another job, we might need to focus on fast retrieval. It is interesting to note that problems that work on the data in parallel are not well suited for linear linked lists but do great with arrays or array of linked data structures. Or, if we wanted to focus on fast retrieval then we would not want to work with an unsorted array (or linear linked list).

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 technical interview problems (e.g., parallel programming, fast searching, etc.). Create a base class for general technical interview questions. This would be an abstract base class since we would never work with a general type of problem.

Then, create three different derived classes for the specific types of technical interview questions. One should be problems that are best suited to parallel programming. The other two are of your own creation. Each type of question should have similarities with each other but at least one should have some different type of functionality (e.g., parallel programming could have the number of threads expected). For all of them you would need a subject, information about the question, the type of data structures that they work best with and or algorithms needed to solve the problem, and the level of difficulty.

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 but the implementation of the methods would need to be different in some way for you to make the most of this experience. In this case, for all types of technical interview questions, we should be able to set a question, edit a question, change the category of question (maybe it doesn't suit parallel programming after all), and rate the difficulty of the question. **Make sure to find at least one method that is different (in name and/or argument lists) so that you can experience how to resolve such differences with RTTI.**

Program #2 – Data Structure Requirements

There needs to be ONE data structure of base class pointers. This data structure will keep all technical questions of all types. In fact, a node should be able to point to any of the three different kinds of questions using upcasting. The data structure will be an Array of Doubly Linked Lists of base class pointers, organized (sorted) by question subject. Then using upcasting, each node can point to the appropriate type of technical question. Implementation of the required data structure(s) requires full support of insert, removal, display, retrieval, and remove-all. There should be a way to be able to display just those technical interview questions of a particular type (e.g., just display those that relate to parallel programming).

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
6. It is expected that you will experience RTTI with dynamic_cast in this assignment

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